UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

REALITY COMPUTING

João Manuel de Castro Afonso

Mestrado em Engenharia Informática

2008

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

REALITY COMPUTING

João Manuel de Castro Afonso

ESTÁGIO

Projecto orientado pela Prof. Dr. Maria Beatriz Duarte Pereira do Carmo
e co-orientado pelo Licenciado em Engenharia Nuno Alexandre Almeida de Sousa Capeta

Mestrado em Engenharia Informática

2008

Declaração

João Manuel de Castro Afonso, aluno nº 30334 da Faculdade de Ciências da Universidade de Lisboa, declara ceder os seus direitos de cópia sobre o seu Relatório de Projecto em Engenharia Informática, intitulado "Reality Computing" , realizado no ano lectivo de 2007/2008 à Faculdade de Ciências da Universidade de Lisboa para o efeito de arquivo e consulta nas suas bibliotecas e publicação do mesmo em formato electrónico na Internet.
FCUL, 28-09-2008

Maria Beatriz Duarte Pereira do Carmo, supervisor do projecto de *João Manuel de Castro Afonso*, aluno da Faculdade de Ciências da Universidade de Lisboa, declara concordar com a divulgação do Relatório do Projecto em Engenharia Informática, intitulado "Reality Computing".
Lisboa, 28-09-2008

Resumo

Este documento descreve o trabalho realizado no âmbito da disciplina de Projecto em Engenharia Informática do Mestrado em Engenharia Informática da Faculdade de Ciências da Universidade de Lisboa.

O trabalho desenvolvido tenta explorar o conceito de reality computing, que visa usar paradigmas de interacção que se relacionam com o natural e o próximo, de forma a obter formas de interacção através das quais é possível tornar os interfaces invisíveis, imediatas e funcionais.

Este é uma abordagem diferente no desenvolvimento de aplicações actualmente, ao contrário do paradigma do *desktop* em que um único utilizador utiliza um único dispositivo para um objectivo especializado, o utilizador pode utilizar simultaneamente vários dispositivos e sistemas, e pode não necessariamente estar a perceber que o está a fazer.

Associado ao conceito de reality computing está, inevitavelmente, um uso intenso da tecnologia que permita não só a detecção da interacção e resposta a esta de uma maneira fluida, perceptível e simplificada, mas também que a comunicação entre estes serviços seja modular e adaptada ao contexto da aplicação.

Desenvolveram-se neste projecto aplicações que ilustram o conceito de reality computing. Estas tiveram por base a framework .Net da Microsoft utilizando e actualizando uma framework feita pela YDreams, o YVision, desenvolvida sobre a versão 3.5 da plataforma da Microsoft na linguagem C#, integrando também varias bibliotecas externas tais como o OpenCv ou o Ogre.

PALAVRAS-CHAVE:

Reality computing, interface pessoa-máquina, realidade virtual, simulação de física, processamento de imagem, realidade aumentada, motor gráfico, computação ubíqua.

Abstract

The work developed tries to explore the concept of reality computing, that aims at using interaction paradigms that relate to the natural and close, in a way that it can make the interfaces invisible, imediate and functional.

This is a different approach to nowadays applications, as opposed to the desktop paradigm, in which a single user consciously engages a single device for a specialized purpose, a user can engage many computational devices and systems simultaneously, in the course of ordinary activities, and may not necessarily even be aware that they are doing so.

Associated with this concept is an intense use of technology that allows that not only the interaction detection and response be held in a fluid, perceptable and simplified way, but also that the communication between these services be modular and adapted to the context of the application.

The development is made under the Microsoft .Net Framework, utilizing and updating a YDreams Framework called YVision, developed under the 3.5 version of the Microsoft platform in the C# programming language, integrating also some external libraries like OpenCv or Ogre.

KEYWORDS:
Reality computing, human-computer interaction, virtual reality, physics simulation, augmented reality, image processing, 3d rendering, ubiquitous computing.

Conteúdo

Lista de Figuras

1 - Introdução

"The most profound technologies are those that disappear. Only when things disappear in this way are we free to use them without thinking and so to focus beyond them on new goals"

Mark Weiser, "The world is not a desktop", ACM publications, 1993.

Uma boa ferramenta é uma ferramenta invisível. Por invisível entende-se que esta não interfere com a nossa consciência, a atenção é focada na tarefa, não na ferramenta. Os óculos são uma boa ferramenta – olhamos para o mundo, não para a ferramenta. Um cego a sondar o terreno com uma bengala sente o terreno, não a bengala. Obviamente que as ferramentas não são invisíveis por si, mas sim inseridas num contexto de utilização. A imersão dos humanos na aplicação, o remover de interfaces mecânicas com o computador (e.g., ratos e teclados), propiciam o desenvolvimento de aplicações com um indicie alto de "invisibilidade", fazendo com que estas sejam actualmente um importante tema de investigação no qual se combinam contribuições de várias disciplinas, nomeadamente, computação gráfica, física, psicologia, ciências sociais e design gráfico.

Este relatório insere-se no âmbito da disciplina de Projecto de Engenharia Informática (PEI) que é parte central do Mestrado em Engenharia Informática (MEI), leccionado pela Faculdade de Ciências da Universidade de Lisboa.

A instituição de acolhimento foi a YDreams que opera no conceito de *reality computing* onde explora a computação ubíqua, o principal objectivo das aplicações descritas neste relatório.

1.1 Motivação

O Desenvolvimento da plataforma YVision permitiu à empresa YDreams actualizar os seus produtos e assim abandonar algumas soluções proprietárias e menos eficazes com que implementava as suas aplicações e reimplementar estas com uma arquitectura renovada que permitisse à empresa melhorar a qualidade geral das suas aplicações. Desta forma surgiu a oportunidade de reestruturar alguns produtos da empresa, assim como desenvolver novos, tirando partido das novas tecnologias integradas nessa plataforma e podendo assim explorar novas oportunidades de levar o conceito de *reality computing* mais longe.

1.2 Objectivos

O conceito de *Reality Computing* vai ser explorado com 3 projectos distintos, para os quais foi necessária a obtenção de complexos conhecimentos gráficos, desde motores de geração de imagens 3D, a motores de simulação de física e bibliotecas de tratamento de imagem. Este tipo de tecnologias tem uma curva de aprendizagem alta, e bastantes paradigmas novos foram estudados para desenvolver estes projectos. O processo de aprendizagem acabou por ser a tarefa que mais tempo consumiu. As responsabilidades do autor no desenvolvimento destas aplicações foram para além das decisões técnicas, as decisões conceptuais da aplicação com vista a melhorar a experiência de utilização sobre o conceito inicialmente proposto. Para além disso, estas aplicações foram desenvolvidas apenas pelo autor, com apoio da equipa de investigação e de design da YDreams, sendo o autor responsável por todo o processo de desenvolvimento, desde a análise de requisitos, a análise de riscos, o controlo de versões, documentação e o suporte à aplicação quando este chegou ao fim do seu ciclo de desenvolvimento.
As 3 aplicações desenvolvidas foram as seguintes:

- Um livro virtual onde a interacção tem de assemelhar o mais possível à realizada com um livro real.
- Um simulador de neve virtual.
- Um Byte mirror, uma ideia de um espelho onde a imagem captada por uma câmara é mostrada, misturada com uma imagem ou vídeo, e onde o movimento destapa uma segunda imagem ou vídeo.

O autor participou também na extensão da plataforma YVision com a construção de diversos blocos que serão adicionados à versão oficial desta.

1.3 Organização do documento

Este documento está organizado da seguinte forma:

- Capitulo 2 – YDreams e a Plataforma YVision
- Capitulo 3 – Frameworks
- Capítulo 4 – Magic Book
- Capítulo 5 – Simulador de Neve
- Capítulo 6 – Byte Mirror
- Capítulo 7 - Conclusão

2 – YDreams e a plataforma YVision

A YDreams é uma empresa Portuguesa fundada em 2000 e que opera essencialmente na área da interactividade. Conta com 150 colaboradores em Portugal e 200 espalhados pelo mundo, com escritórios em Barcelona, Xangai, Brasil, Estados Unidos e Reino Unido. As principais aplicações desenvolvidas pela YDreams são:

- Aplicações em sistema de informação geográfica.
- Interfaces baseadas em sensores de movimento.
- Biometria.
- Processamento de imagem
- Soluções em realidade aumentada.

A plataforma YVision desenvolvida no YLabs, o departamento de investigação da YDreams, é um agregador de *frameworks* desenvolvida com vista à produtividade e reutilização de código. A plataforma é revolucionária principalmente ao nível da gestão de threads e da integração de ferramentas.

2.1 Problemas

O YVision é um integrador de tecnologia, destinado a resolver as questões que a YDreams tinha ao utilizar as diversas *frameworks* independentemente. Estas questões eram principalmente:

- Reutilização e partilha de código complexa e difícil integração
- Personalização das aplicações muito dispendiosa
- Problemas com as versões das framework, que aumentava a complexidade do controlo de qualidade
- Calibração complexa das aplicações e documentação incompleta, o que provocava que a manutenção dos projectos se tornasse muito dispendiosa.

2.2 Arquitectura

Face às questões anteriormente mencionadas o YVision tem uma arquitectura que facilita a inovação e a reutilização do conhecimento, ao mesmo tempo que permite uma dramática redução no tempo de desenvolvimento e personalização das aplicações. Devido também ao seu robusto e eficaz sistema de threading permite criar aplicações de desempenho elevado e aumentar a robustez dos produtos produzidos.

A plataforma íntegra:

Processamento de imagem
Efeitos de vídeo
3D rendering
Simulação de física
Áudio 2D e 3D
Camera Tracking

A arquitectura de desenvolvimento do YVision está dividida em 2 conceitos essenciais:

- Grafos

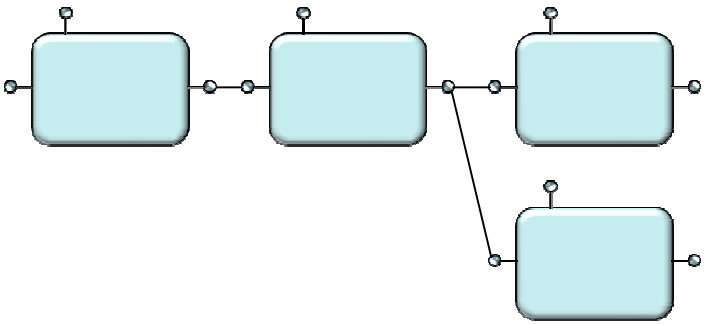
- Objectos

2.2.1 Grafos

Para situações em que os dados são efémeros, (e.g. sequencias de tratamentos de imagem, analise de dados de uma câmara), a plataforma YVision utiliza uma abstracção de blocos e ligações de fluxos entre eles.

2.2.1.1 Blocos

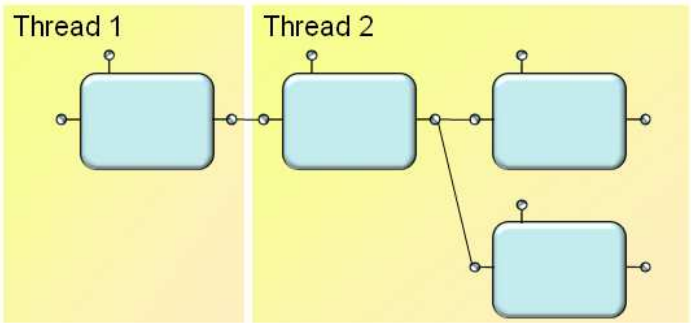
Um bloco é um encapsulamento "black-box" de alguma tarefa útil. Estes podem-se ligar entre si criando um grafo (figura 1). Os fluxos entre os blocos são dinâmicos e suportam alterações em tempo de execução, junções de 2 ou mais fluxos e sinalizações. Os fluxos são no entanto abstractos o suficiente para permitir que transaccionem qualquer tipo de dados.



1. Grafo de Blocos

2.2.1.2 Threads

Cada bloco é executado num contexto sincronizado que automaticamente direcciona a propagação de eventos para as diferentes *threads* do YVision (Figura 2), criando uma abstracção para o programador do sistema de threads na verdade é executado pelo sistema operativo.



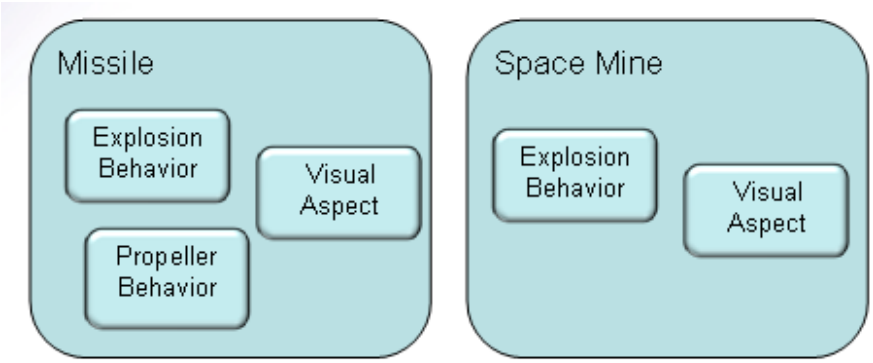
2. Gestão de Threads
diferentes blocos podem correr em diferente threads mas toda a gestão de comunicação e sincronização é efectuada pela plataforma

2.2.1.3 Conclusão

Esta organização em grafo é principalmente útil em termos de reutilização de código. Todos os dados do tipo efémero ao serem correctamente implementados nesta abstracção são facilmente reutilizados por outros programadores, com uma necessidade de documentação reduzida, aumentando assim a produtividade e cooperação entre equipas diferentes. O inovador sistema de *threading* ao permitir que cada bloco corra num contexto próprio permite níveis de desempenho bastante optimizados, pensando não só nas vantagens do processamento paralelo numa só unidade de processamento mas tirando já partido das actuais arquitecturas dual core, onde maior porta das aplicações da YDreams já corre.

2.2.2. Objectos

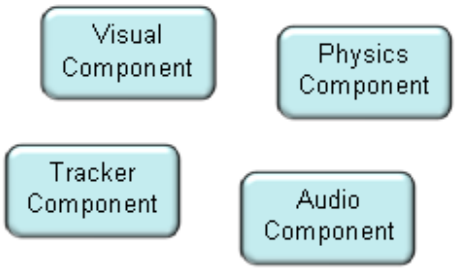
Os blocos e as ligações entre eles tratam fluxos de dados efémeros. Este paradigma é, no entanto, inadequado para tratamento de tipos de dados persistentes. Algumas aplicações têm a necessidade de definir elementos virtuais (as "coisas" que persistem no mundo virtual). A arquitectura do YVision vê este tipo de dados como objectos compostos por componentes (Figura 3). Esta arquitectura tem a vantagem de ser muito flexível e muito modular, não forçando hierarquias ou contextualizações pouco dinâmicas. Numa arquitectura definida por componentes um objecto não é definido por herança, mas sim por composição de funcionalidades e dados.



3. Objectos definidos por composição
Neste exemplo podemos observar como os objectos Missile e Space Mine são compostos por um conjunto de componentes

2.2.2.1 Componentes

Um componente implementa um conjunto específico de funcionalidades e dados úteis (Figura 4), definidos por uma interface. Os componentes são organizados por famílias de funcionalidades. Um objecto não pode ter mais do que um componente da mesma família.



4. Componentes
Exemplos de componentes que constituem os objectos, podemos verificar neste exemplo que este objecto vai ter uma componente visual, correspondente à sua representação visual no mundo gráfico, uma componente de física correspondente à sua representação no mundo da simulação de física, etc.

2.2.2.2 Conclusão

Os objectos permitem ter a parte persistente da aplicação de forma modular, flexível e principalmente personalizável. A definição dos componentes que constituem um objecto é configurável sempre externamente à aplicação e esta independência permite alterar todo o comportamento e aspecto de uma aplicação sem obrigar a uma alteração de código.

3 - Frameworks

Nesta secção iram ser descritas as bibliotecas que foram estudadas e usadas para a realização das aplicações. Todas estas bibliotecas são ou de código aberto ou de utilização livre, são na sua maioria escritas em linguagens unmanaged (C, C++ principalmente), mas *wrappers* (encapsulamentos das funções noutras linguagens) foram usados ou desenvolvidos para ser possível a sua utilização a partir da linguagem C#

3.1 OpenCV

O OpenCV (Open Source Computer *Vision*) é uma biblioteca de programação orientada principalmente para o processamento de imagens em tempo real desenvolvida pela Intel. As principais utilizações são no campo das *Human Computer Interactions*, identificação de objectos, reconhecimento e segmentação, reconhecimento de caras, reconhecimento de gestos e movimentos, compreensão do movimento e estruturas do movimento. È uma biblioteca aberta e está disponível em http://opencvlibrary.sourceforge.net.

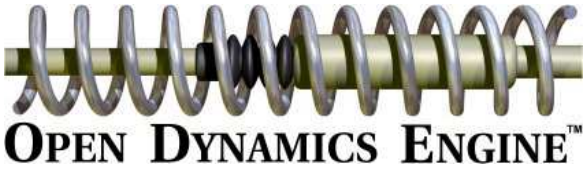
As funções desta biblioteca foram elementos chave para a realização dos projectos aqui descritos, foi necessário obter vários tipos de funcionalidades de processamento de imagem, dentro da categoria de detecção e análise de movimento e detecção de contornos.
Para encontrar os contornos existentes numa imagem foi usado o método *cvFindContours*, no entanto este método apenas funciona sobre imagens com um único canal de 8 bits binário, ou seja todos os pixéis com cor eram tratados como 1 e todos os sem cor como 0 que obriga à conversão das imagens mas com outro filtro o BGR2BW.
Para obtenção a velocidade do movimento na câmara a função *CalcOpticalFlowPyrLK* que calcula os vectores de movimento entre duas imagens utilizando o método iterativo *Lucas-Kanade* em pirâmides foi a escolhida como algoritmo *de optical flow* devido ao seu bom desempenho ao nível da detecção de ruído.
Os algoritmos de *Optical Flow* estimam as deformações entre duas imagens e calculam os vectores de movimento entre os pixéis destas duas permitindo assim descobrir a velocidade a que os objectos na câmara se estão a mover.
Esta função tem algumas pré-condições para trabalhar com eficiência, sendo uma delas a passagem das imagens recebidas para escala de cinzentos (utilizando o *color* converter do *OpenCV* com o filtro BGR2GRAY).
Como acumuladores de imagem, funções que vão somando as imagens fornecidas, foram usadas as funções *cvAcc* e *updateMotionHistory.* A função *cvAcc* do *OpenCv, faz* parte de uma categoria designada de análise de movimento e permite somar imagens de maneira a acumulá-las ao longo do tempo. A função *updateMotionHistory* faz parte da mesma categoria da anterior, mas nesta a relação com o tempo é diferente. À medida que são adicionadas imagens a esta função apenas os pixéis mais recentes são deixados no resultado final, e as imagens adicionadas à mais tempo vão progressivamente desaparecendo.

3.2 Ogre 3D

5. Logotipo do Ogre
(Retirado de http://www.ogre3d.org)

O OGRE (Object-*Oriented Graphics Rendering Engine*) é um motor de *render* 3D flexível (e não um motor de jogo). Escrito em C++ e desenhado para tornar mais fácil e intuitivo para os programadores produzirem aplicações utilizando gráficos 3D que tiram partido da placa gráfica. A Biblioteca abstrai os detalhes de utilização das bibliotecas de suporte como Direct3D ou o OpenGL.

3.3 ODE



6. Logotipo do ODE
(Retirado de http://www.ode.org)

O ODE (Open Dynamics Engine) é uma biblioteca de alto desempenho na simulação da dinâmica de corpos rígidos. Contem tipos de junções avançados, e detecção de colisões integrada com fricção. O ODE é útil para a simulação de veículos, objectos num ambiente de realidade virtual e de criaturas virtuais. É actualmente usado em muitos jogos de computador, ferramentas 3D e ferramentas de simulação.

O ODE possui o conceito (partilhado também pela arquitectura de componentes do YVision) de *World*, que neste contexto se refere a um contentor de corpos rígidos (bodies) e de junções (joints). Objectos que residam em mundos diferentes não interagem e todos os objectos que façam parte de um mundo existem no mesmo ponto no tempo.

No ODE uma simulação típica é executada com o seguinte fluxo:

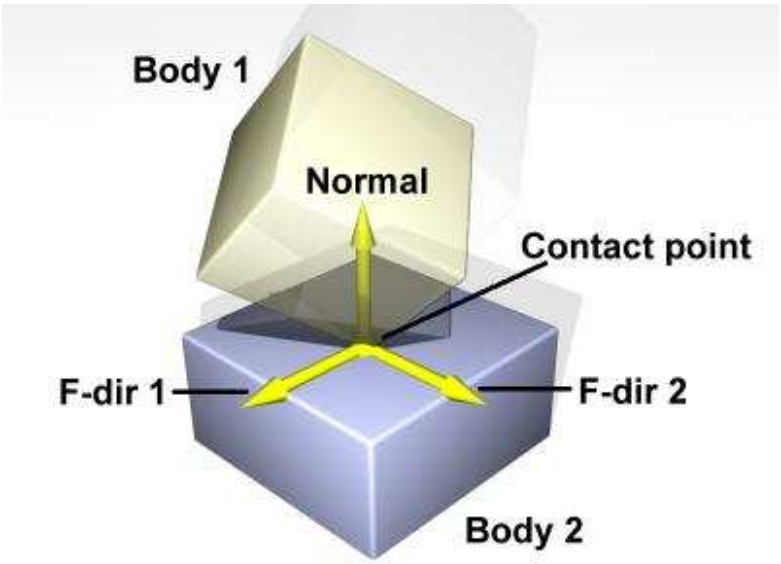
- Criação de um *world* dinâmico
- Criação dos corpos no mundo dinâmico criado
- Configuração dos estados (posição, forças,..) de todos os corpos
- Criação de todas as junções existentes

- Acoplamento das junções aos corpos correspondentes
- Configuração das junções criadas
- Criação de um world para a colisão apenas com as geometrias dos corpos
- Criação de um grupo de junções que irá conter todos os contactos entre corpos
- Ciclo:
  - Aplicar as forças aos corpos como necessário
  - Ajustar os parâmetros das junções como necessário
  - Fazer a chamada à detecção de colisões
  - Criar uma junção de contacto por cada ponto de colisão e adicioná-la ao grupo de junções dos contactos
  - Executar um passo da simulação
  - Remover todas as junções adicionadas ao grupo de junções dos contactos
- Destruição de todos os grupos de contacto

As colisões entre corpos e entre corpos e o ambiente estático à volta são geridas da seguinte maneira

- Como referido acima antes de cada passo da simulação, deve ser efectuada uma chamada às funções de detecção de colisão para determinar que corpos estão a tocar em quê. Estas funções retornam uma lista de pontos de contacto, cada um destes pontos especifica uma posição no espaço, um vector normal à superfície e a profundidade de penetração no corpo.
- Uma junção especial de contacto (Figura 7) é criada para cada ponto de contacto detectado. Cada uma destas junções de contacto contém informação adicional sobre o contacto, como por exemplo a fricção presente na superfície de contacto ou a elasticidade e rigidez da mesma.
- Estas junções de contacto são colocadas num grupo próprio o que permite que elas sejam colocadas e retiradas com muita velocidade, este é um ponto importante pois a velocidade da simulação decresce quando o número de contactos sobe. É portanto importante determinar estratégias com vista a limitar o número de pontos de contacto.
- È executado o passo da simulação.
- São removidos todos os pontos de contacto do sistema.

Junções de Contacto



7. Junções de Contacto
(retirado de http://www.ode.com)

As junções de contacto impedem o corpo 1 e o corpo 2 (Figura 7) de se penetrarem mutuamente no ponto de contacto, ao permitir aos corpos que tenham uma velocidade de saída na direcção da normal ao ponto de contacto, estas junções como já foi referido têm tipicamente um tempo de vida de um passo da simulação. Elas são criadas e apagadas em resposta à detecção de colisão.

Definição das características de uma superfície

A reacção desencadeada por uma colisão é configurada pelas características das superfícies dos corpos em contacto. Quando uma *contact joint* é criada uma estrutura do tipo *dContact* é fornecida ao sistema. Esta tem a seguinte definição:

```
struct dContact {
  dSurfaceParameters surface;
  dContactGeom geom;
  dVector3 fdir1;
};
```

8. Estrutura do contacto do ODE (retirado de http://www.ode.com)

fdir1 é um vector que indica a "first friction direction" que define a direcção aplicada a cada força de fricção, poderá existir uma "second friction direction", mas esta é computada como sendo um vector perpendicular entre a normal do ponto de contacto e a fdir1. Geom define a geometria do contacto, e a surface é uma sub estrutura definida pelo utilizador que define as características da superfície. É esta propriedade que vai definir a reacção que os corpos vão ter na altura da colisão. Estas propriedades são mostradas a seguir

Mu	Coeficiente de fricção de Coulomb. Isto necessita de estar no intervalo entre zero e dInfinity. Zero significa uma superfície sem atrito, e dInfinity resulta numa superfície que nunca escorrega. De notar que as superfícies sem atrito consumem menos tempo de computação e que superfícies com atrito infinito (dInfinity) podem ser mais leves de computar que superfícies com atritos finitos.
Mu2	Coeficiente de Coulomb opcional para definir o coeficiente de fricção para a "second friction direction", senão for usada o valor definido para o Mu é usada para as duas direcções de fricção
Bounce	Parâmetro de restituição de força, representa a elasticidade de uma superfície.
Bounce_vel	A velocidade mínima necessária para as superfícies para o parâmetro Bounce iniciar a sua acção. Corpos que colidam com uma velocidade inferior a este parâmetro nesta superfície terão um valor de bounce de 0.
Soft_ERP	Controla o Error Reduction Parameter que permite

	algum erro de posicionamento entre dois corpos unidos por uma junção.
Soft_CFM	Controla o Constraint Force Mixing, que indica o rigor com que as restrições são aplicadas aos corpos
Motion1, motion2	Quando estas variáveis são usadas é assumido que a superfície tem uma velocidade independente do movimento dos corpos, com estas variáveis é possível definir as velocidades de atrito independentes do movimento do corpo
Slip1, Slip2	Os Coeficientes de force-dependent-slip (FDS) para as direcções 1 e 2 de fricção.

3.4 - Integração das Frameworks – Ogre- Ode-OpenCv

As 3 frameworks usadas nestes projectos dispõem obviamente cada uma de um contexto próprio, e tem de ser actualizadas e convertidas de maneira a poderem interagir entre elas. A integração entre o *ODE* e *Ogre* é feita através de um *Updater Component* (explicação mais à frente) que irá actualizar a posição e orientação, o passo de execução da física é actualizado a cada *frame* rendido.
A integração do OpenCv com as restantes frameworks é mais complicada. Um caso complicado é a possibilidade de mostrar uma imagem vinda do *OpenCV* (A YDreams possui um *wrapper* proprietário desta *Framework* que transforma o OpenCv numa *framework* orientada a objectos, pelo que as imagens provenientes desta Frameworks são conhecidas com *CvImages*) no *Ogre*. A solução encontrada pelo YVision foi a de desenvolver um bloco que designou com *CvImageToTexture* que converte uma *CvImage* numa textura passível de ser aplicada a qualquer mesh e de sofrer pós processamento na placa gráfica. Este bloco permite actualização muito velozes das texturas sendo possível converter para texturas, streams provenientes do bloco de Câmara com *frame rate* constante e elevado.

Outro assunto complicado é integração entre o *OpenCV* e o *ODE*. A ideia é que seja possível dar propriedades de surface de *ODE* aos resultados dos cálculos de contornos e de *optical flow* de maneira a este poder gerir a colisão entre estes e os objectos visuais que irão aparecer no ecrã (a neve) que também terão propriedades de surface próprias. Para que tal aconteça é necessário ligar a cada bloco resultante um outro bloco que irá transformar a estruturas tipo grafo resultantes do *OpenCv* em meshes triangulares que poderão ser registadas no mundo do *ODE*, isto é feito através da criação de um plano a uma distância física e fazendo a proporção entre as coordenadas dadas pelo *OpenCV* e as coordenadas nesse plano. Todas estas meshes produzidas irão ter a mesma profundidade e distância ao *Point of View* (normalmente será a câmara do Ogre), mas a sua altura e comprimento irão variar consoante o *input* fornecido. A estas geometrias já irá ser possível dar propriedades de superfície, que irão efectuar as colisões consoante os parâmetros fornecidos.

4 - Magic Book

Aqui é descrito o processo de evolução da 1ª aplicação efectuada para a YDreams, o Magic Book, o conceito do produto, o processo de análise de requisitos efectuado e a arquitectura desenhada a partir destes, screenshots demonstrativos da aplicação e uma reflexão sobre este primeiro projecto.

4.1 Motivação

O Magic Book é um produto antigo da YDreams, implementado numa plataforma proprietária chamada Virtools (http://www.virtools.com/). Neste primeiro projecto a aplicação existente foi transportada para a plataforma YVision, além disso melhorou-se a gestão de memória da aplicação.

4.2 Conceito

O Magic Book é um livro 3D dinâmico, que suporta diferentes tipos de média: vídeos, imagens e sons. A interacção é feita através de uma barra de sensores que activa o rolhamento das páginas do livro, tentando assim retratar a realidade, explorando o conceito de *reality computing*.

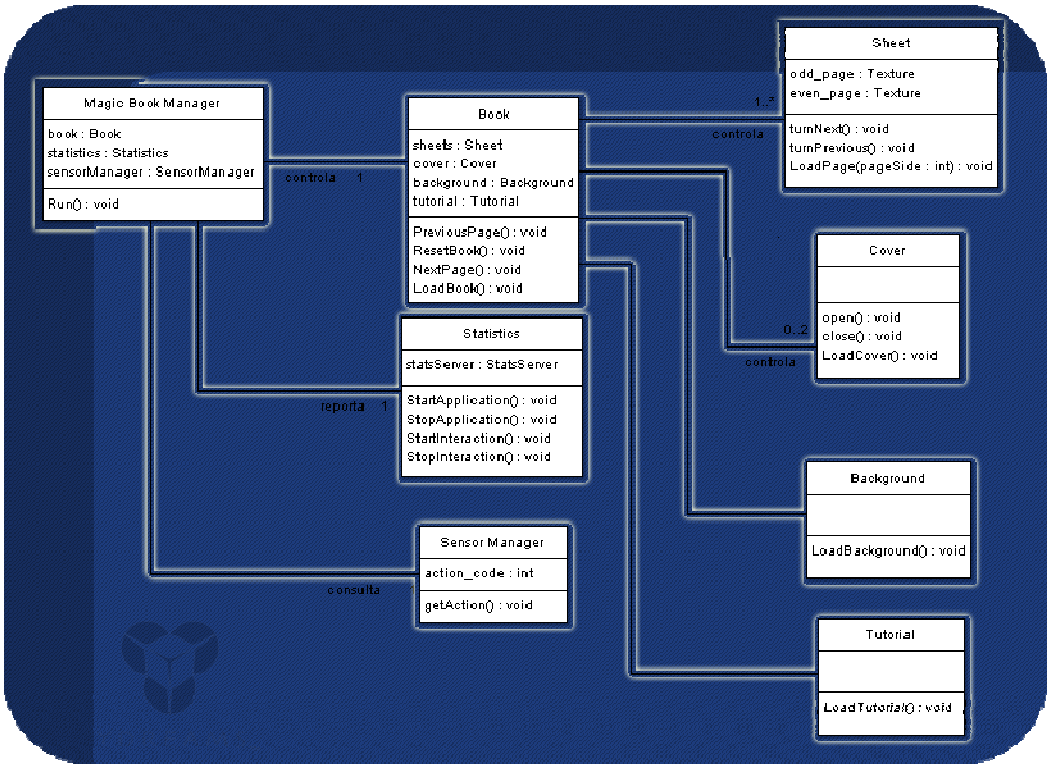
4.3 Requisitos

A análise de requisitos foi baseada na análise dos requisitos que a versão antiga dispunha, combinada com uma análise das queixas que os utilizadores tinham dessa versão. Além disso, teve-se em atenção uma análise das prováveis funcionalidades que este irá ter de futuro, de maneira a poder criar uma arquitectura que fosse modular o suficiente para permitir a adição destas. O método de recolha destes requisitos usado foi uma análise às funcionalidades mais usadas da versão antiga do livro e depois de várias reuniões com o departamento de design em busca de novas funcionalidades que seria interessante o livro ter, mesmo que não fosse para esta versão inicial.
A lista dos principais requisitos é mostrada de seguida:

Simulação de um livro em 3d
O conteúdo das páginas é fornecido externamente e pode ser uma imagem estática ou um vídeo
A interacção é feita através da comunicação e interpretação dos dados de um sensor de movimento
A animação de virar de página poderá ter um som associado
Simulação de Luzes
Gestão eficaz de memória
Possibilidade de inclusão de vídeos de ajuda à utilização

Registo de estatísticas de utilização
Permitir o reset do livro passado um tempo configurável sem interacção
Permitir a utilização de 4 tipos de câmaras diferentes – full screen, flipped, perspective e side *camera*
Suportar sem distorção monitores 16:9 e 4:3
Preparação para inserção de uma mesh e animação diferente para a capa do livro no futuro
Preparação para inserção de um sistema de animação próprio para quando as folhas estão paradas (idle mode)

4.4 Arquitectura



9. Arquitectura do Magic Book

A figura 9 apresenta o modelo desenhado para a implementação do magic book. O design pattern Model View Controller foi utilizado para uma melhor separação das responsabilidades, mas não implementado à risca. Estas questões assim como a descrição das classes vão ser apresentada de seguida. No *design pattern Model View Controller,* o *model* representa a informação (os dados) da aplicação e lógica usada para manipular essa informação, a *view* corresponde aos elementos visuais da interface e o *controller* gere as comunicações entre os dois.

4.4.1 Descrição das classes

MagicBookManager:
Esta é o ponto de entrada da aplicação, e também o *controller*. Tem as seguintes responsabilidades:

- Inicializa o motor de *render* e de todas as classes do nível abaixo.
- Controla o ciclo de *render* e actualiza todas as acções *time-aware* que estiverem a decorrer.
- Reportar eventos de interacção para a instância da classe *Statistics.*
- Tratar o input vindo do objecto da classe *SensorManager* e notificar as classes certas.

Statistics:
Todas as aplicações da YDreams têm de reportar estatísticas sobre a sua utilização. Os relatórios são depois enviados para um servidor, mas é esta classe que tem a responsabilidade de saber o que fazer aos eventos que lhe são reportados pelo controller (Classe *MagicBookManager*)

Sensor Manager:
A leitura do sensor é assíncrona por questões de desempenho. É este *manager* que tem a responsabilidade de ler os dados do sensor, de maneira a informar correctamente o controller de que uma interacção foi desencadeada.

Book:
Este é o gestor do livro, e também o *model,* e como model é ele que tem o estado da aplicação. As responsabilidades desta classe são as seguintes:

Gerir a memória, é esta classe que tem de alocar e carregar dinamicamente os recursos necessários

Gestão das views, é esta classe que sabe qual elemento visual (*folha, capa*) a actualizar

Gestão das profundidades, esta responsabilidade será explicada mais tarde

Sheet:
Esta é uma das views, que representa uma folha, com as suas 2 páginas, uma impar e uma par, de cada lado da folha. É esta classe que tem a responsabilidade de suportar os tipos de média pedidos (imagem, vídeo e som) e fazer as animações.

Cover:
Esta é mais uma view, com a mesma responsabilidade da classe *sheet*, mas esta representa a capa do livro.

Background:
Outra *view*, esta representa o fundo. Suporta também imagens ou vídeos.

Tutorial:
Esta *view* representa uma pequena imagem ou vídeo exemplificativo da utilização do livro que poderá aparecer na primeira página, para ajudar os utilizadores a entender a sua forma de interacção.

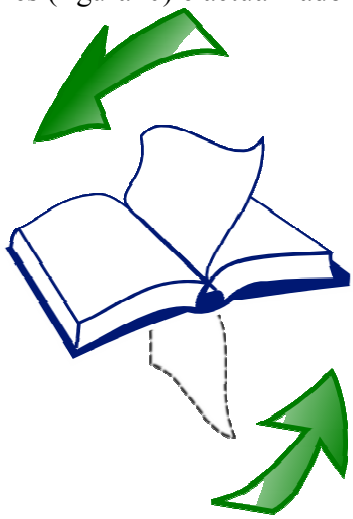
Foram criadas outras classes auxiliares menos relevantes mas em termos de estrutura, foram estas as classes principais. De seguida será descrito o algoritmo de gestão de memória implementado.

4.5 – Gestão de Memória

Visto o livro lidar com tipos muito pesados de media (vídeos principalmente), não era uma boa opção carregar todas as páginas para a memória inicialmente. Obrigatoriamente o carregamento das páginas teria de ser dinâmico. Com vista a resolver este problema foi desenvolvido o seguinte algoritmo para o carregamento dinâmico das páginas.

4.5.1 – Algoritmo de gestão da memória

- São carregadas até 6 *meshes* com os respectivos tipos de média que constituem as página. Este número foi o que melhor servia o propósito de ciclo de páginas que o algoritmo implementa sem haver flickering ou qualquer outro tipo de ruído visual.
- Quando o número total de páginas é superior a 6, quando é chegado ao meio do livro é iniciado um ciclo às meshes (figura10) e actualizado o seu conteúdo



10. Gestão de memória do Magic Book
Esta imagem demonstra o ciclo de meshes que é efectuado de maneira a que sejam sempre as mesmas

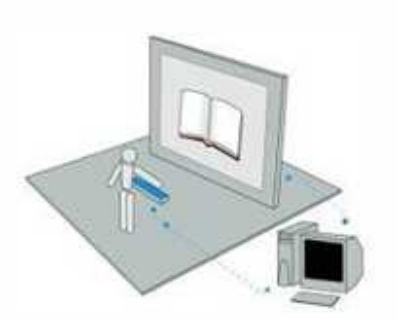
folhas no livro

Assim, por cada página virada, a página de 2 posições atrás é passada 2 posições para a frente (isto para o caso de página estar a ser virada no sentido contrário ao ponteiro dos relógios, o inverso se verifica na situação contrária), criando assim um ciclo de meshes, obviamente que o conteúdo da página depois do ciclo é actualizado e as texturas dessa mesh actualizados com os conteúdos pré-definidos. A inicialização dos vídeos é assíncrona, e portanto o comportamento do livro não é influenciado, enquanto os vídeos são inicializados.

4.5.2 – Conclusão sobre o algoritmo

A questão mais importante aqui é o número inicial de páginas. Seis folhas podem corresponder a 12 vídeos inicializados, mais o tutorial e o background. No entanto, depois de algumas medições efectuadas nesse cenário, concluiu-se que esse peso inicial era suportável. O algoritmo cumpre bem a sua função, foram efectuados testes com 120 páginas todas no formato vídeo e a memória e o processamento mantiveram-se estáveis e a níveis aceitáveis.

4.6 – Interacção com o sensor

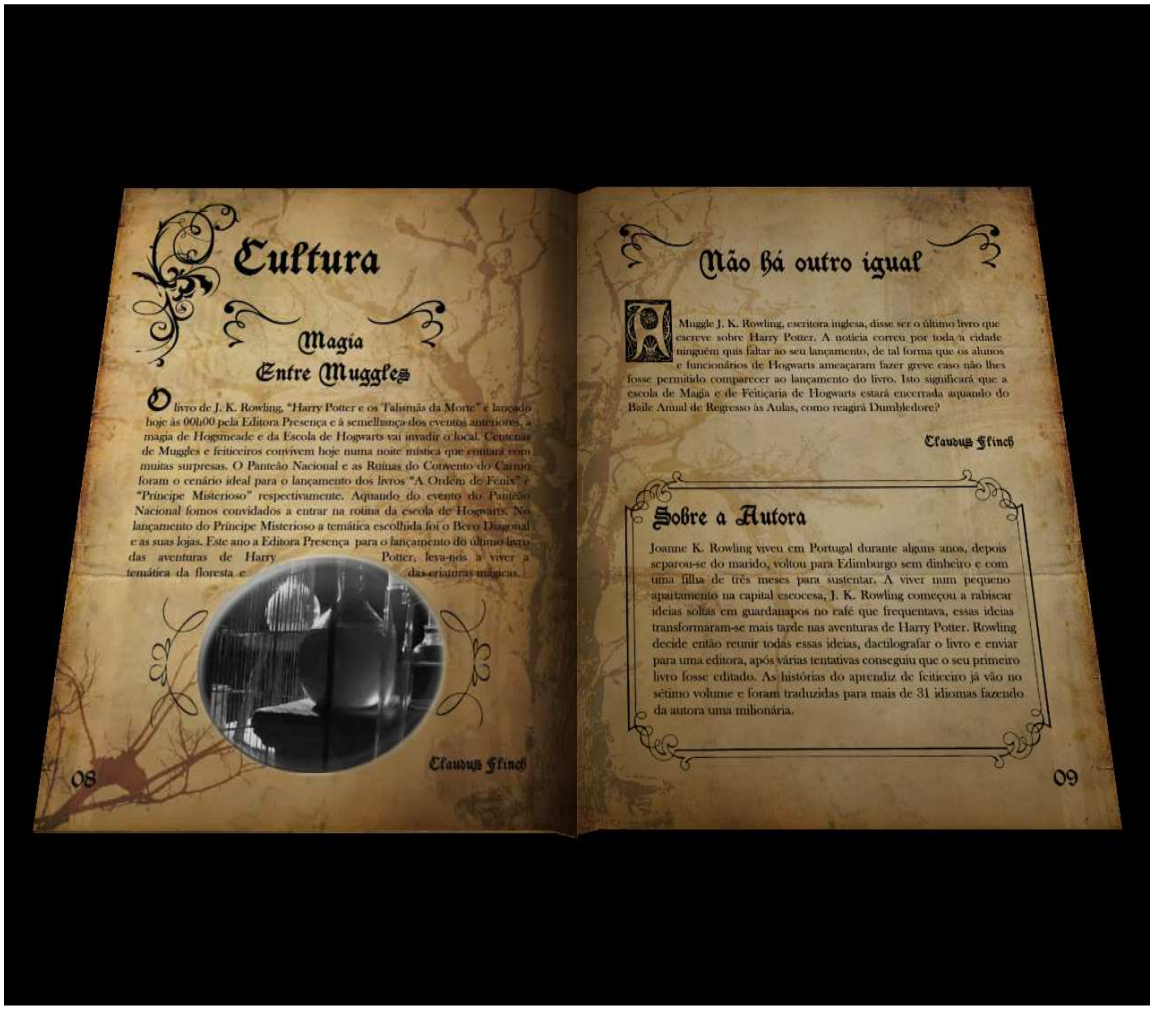


11. Interacção com o sensor

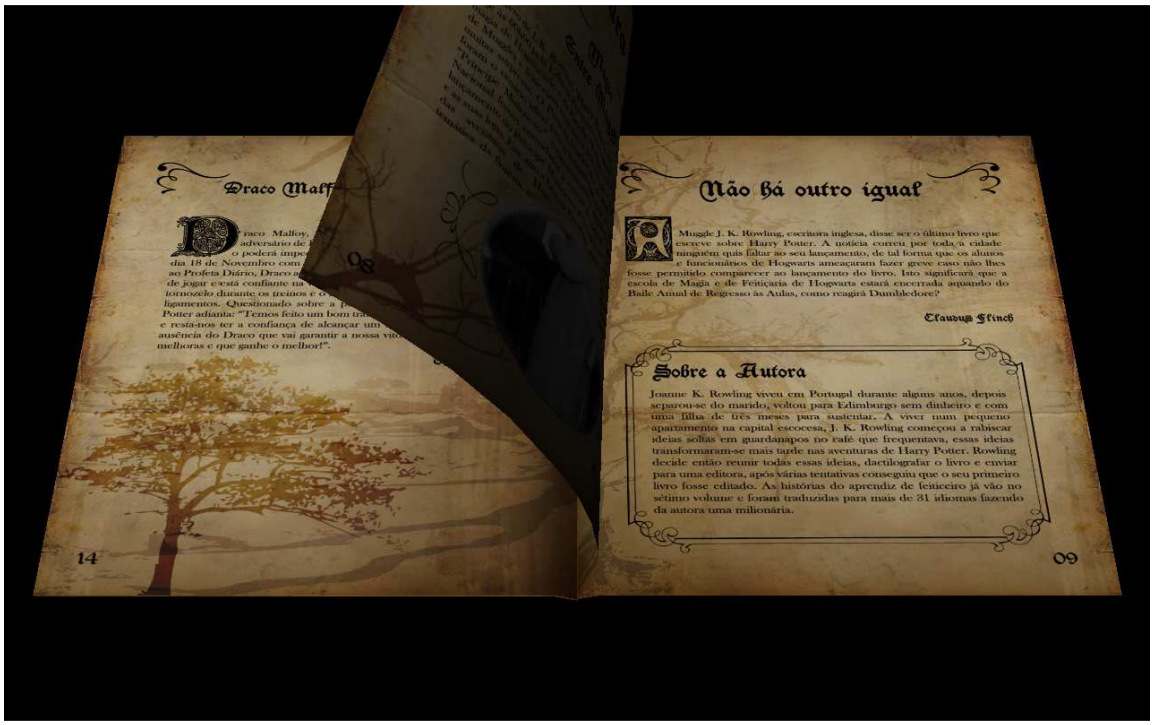
A figura 11 mostra a interacção com o sensor. O sensor envia dados relativos à posição da mão por cima deles. Existe algum ruído nos sensores, e por isso a YDreams desenvolveu uma aplicação que aplica tratamentos estatísticos a esses dados e que dispõe de regras que enviam códigos especiais quando é considerado que a mão atravessou toda a barra de

sensores de uma ponta a outra e se pode então iniciar o virar de uma página e em que direcção.

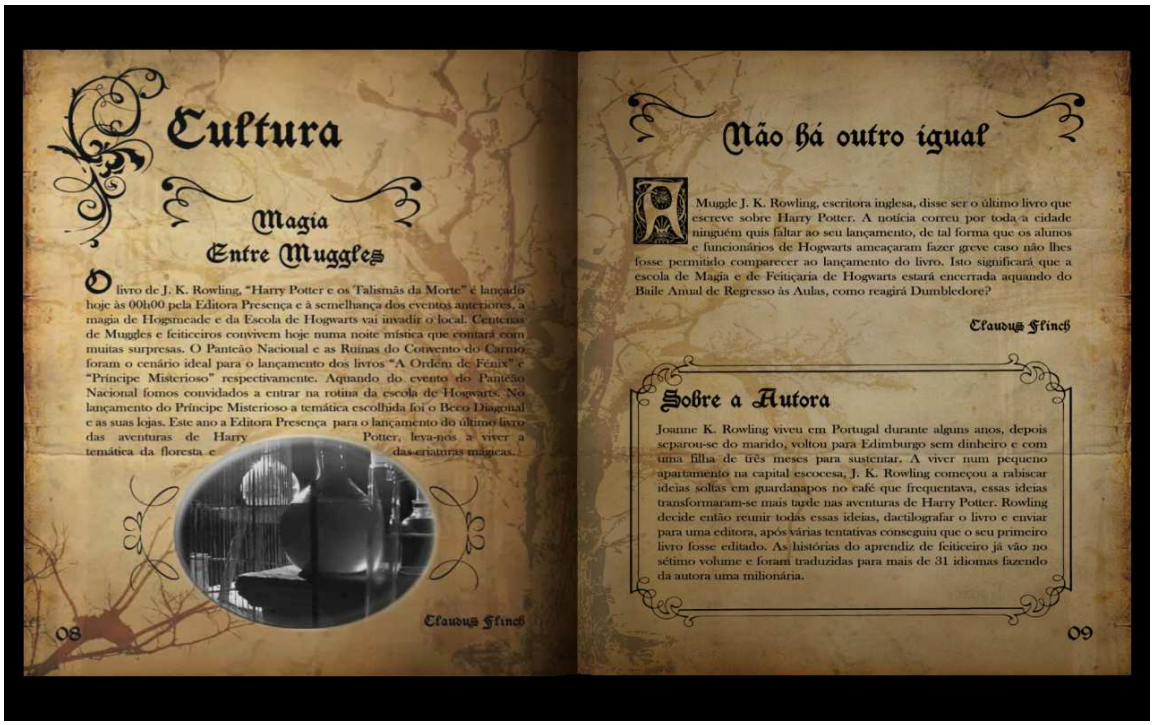
4.7 – Screenshots da Aplicação



12. Screenshot do Magic Book
a correr um vídeo e com a câmara em perspectiva



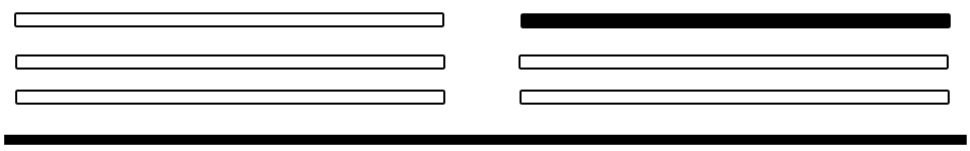
13. Screenshot do Magic Book
no decorrer de uma viragem de uma página onde se pode observar a gestão das sombras e a transparência do algoritmo de gestão de memória



14. Screenshot do Magic Book
a correr um vídeo na câmara superior e podemos observar a gestão das sombras a meio do livro

4.8 – Conclusão

Este projecto deu para perceber muitos dos problemas de tentar computar a realidade o mais perto possível, o maior obstáculo encontrado foi a simulação da colisão entre as páginas. No mundo real uma página só vira até colidir com outra página parando aí o seu movimento. No mundo 3D essa colisão não existe implicitamente e a sua computação ira tornar a aplicação muito mais pesada, sendo uma solução muito violenta para a questão. Um algoritmo de gestão de profundidades foi implementado para resolver esta situação. A profundidade da mesh era corrigida consoante a sua posição no livro (figura 15).



15. Gestor de Profundidades do Magic Book

Foi um problema interessante deste tipo de objectivo de emular a realidade, cuja solução

não se baseou na realidade. Outra questão interessante foi a decisão de o que deveria ser o despoletar da interacção. Na realidade as paginas acompanham as nossas mãos ate ao final do virar da página, mas essa interacção revelou-se confusa, pois muitas vezes as pessoas não tinham a percepção correcta se tinham virado a totalidade da página e acabavam por ficar frustradas.
Foi uma boa introdução ao mundo do reality computing, com bons desafios e a possibilidade de uma entrada mais suave nas tecnologias que a YDreams usa.

5 - YFlakes

Aqui é descrito o processo de evolução da 2ª aplicação, o simulador de neve, explicando desde a motivação ao conceito. A abordagem diferente no que toca à análise de requisitos, uma descrição de como as frameworks foram usadas neste projecto, e por fim arquitectura desenhada. È apresentado no fim do capitulo uma reflexão e conclusão sobre o projecto.

5.1 Motivação

O YFlakes é um simulador de neve com simulação de física. A ideia original seria uma adaptação de um produto da YDreams denominado Bubbles, mas devido a esse produto não ter sido desenvolvido com a totalidade dos paradigmas do YVision (alguns não estavam completamente desenvolvidos na altura do desenvolvimento deste produto), acabou por ser feito de raiz, aproveitando alguns conceitos adquiridos com sucesso nesse projecto.

5.2 Conceito

A ideia deste simulador é a neve colidir com os contornos dos objectos capturados com uma câmara, e acumular-se nesses contornos. A colisão deve respeitar as regras de física, e o efeito de neve deve ser recriado de maneira realista. Este projecto tinha um cliente específico, e portanto, estava sujeito à sua aprovação final, pelo que os objectivos podiam ser sujeitos a ajustes consoante a vontade deste cliente.

5.3 Requisitos

Sendo este projecto para um cliente específico, a abordagem na análise dos requisitos é toma uma perspectiva diferente. Foi estabelecido um canal de comunicação com o cliente com vista a garantir que eram estes os requisitos certos. Depois de algumas reuniões os requisitos que foram identificados foram os seguintes:

Detenção dos contornos das imagens capturadas com a câmara
Recriar das condições de queda da neve
Colisão e acumulação dos flocos de neve

5.4 Frameworks

5.4.1 ODE

Como explicado no capítulo 3.3 ODE, este é o motor de física utilizado no YVision. Para replicar o comportamento físico da neve foi necessário criar um corpo de física para cada floco e parametrizar as suas características de *surface,* facto que se revelou bastante complicado devido à instabilidade que era introduzida nas reacções, provocando várias vezes que os flocos subissem inesperadamente e com grande velocidade.

5.4.2 Open CV

Esta aplicação obrigou também à primeira aproximação ao *OpenCV* a biblioteca de tratamento de imagem da *Intel*, com vista a proceder à detecção dos contornos e vectores de movimento da pessoa. Para a detecção dos movimentos foi usada um algoritmo *de optical flow*, previamente apresentado no capítulo `3.1 OpenCV`. A assumpção principal dos algoritmos de *optical flow* é que os valores de intensidade e cor dos pixéis ou voxels entre as duas imagens não sofreram alterações significativas.

O método *cvFindContours* (apresentado em `3.1 OpenCV`) foi o escolhido para detectar os contornos das imagens, no entanto os contornos apresentados não estavam satisfatórios, pois apanhavam demasiado ruído do *background* que maior parte das vezes era estático e que portanto apresentava como *output* contornos a cobrir o ecrã inteiro o que impossibilitava totalmente a interacção. A maneira de resolver este problema foi através da adição prévia de um bloco anteriormente elaborado pela YDreams conhecido como *DifferenceBlock*, o que este bloco permite é a subtracção do *background* estático, através de um *threshold* configurável que permitiu reduzir o nível de ruído para valores aceitáveis, retirando os elementos que continuavam estáticos no ecrã e permitiu capturar os contornos apenas nos elementos dinâmicos da imagem,

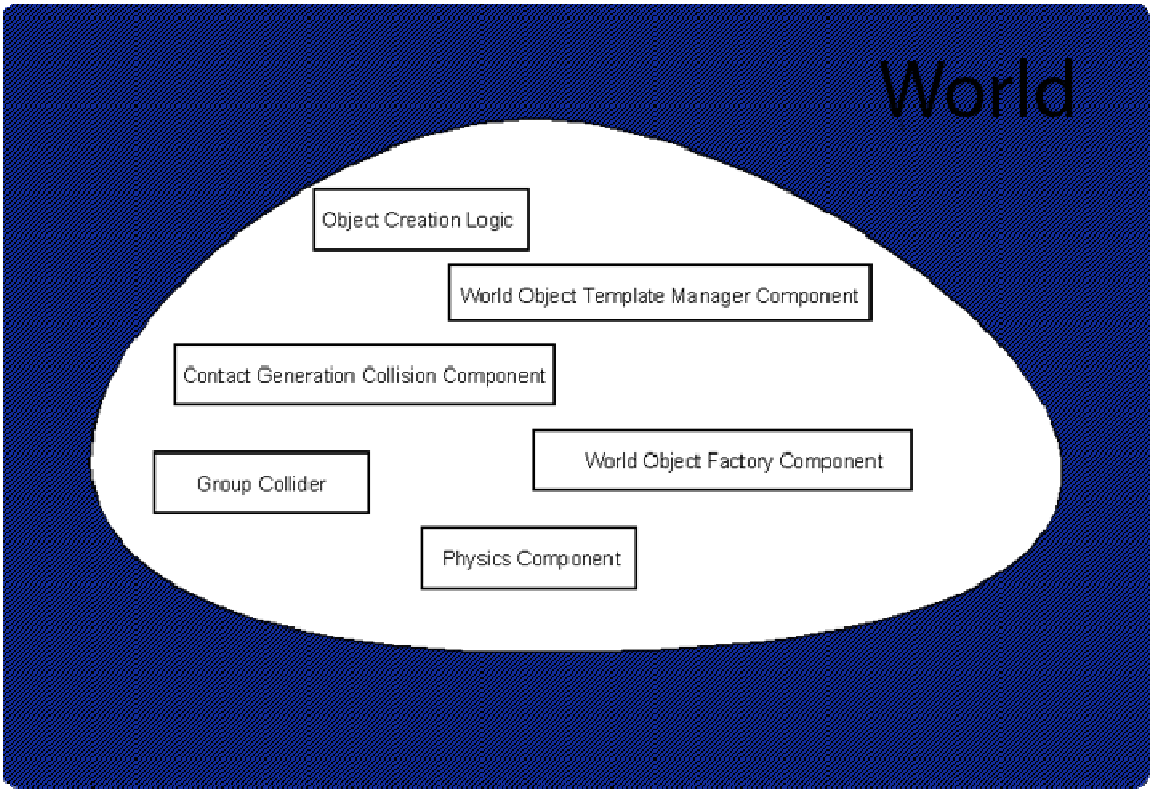
5.5 Arquitectura

Depois de introduzidos os conceitos que foram necessários estudar para esta aplicação passemos então para a arquitectura desenvolvida. Esta aplicação foi a primeira a incluir o conceito dos componentes do YVision anteriormente falado, estes tratam não só da representação visual dos objectos persistentes do ecrã, mas como da sua representação física e actualizadores, é possível também definir componentes de tratamento de colisão. A sua definição por composição é uma boa aposta pois permite a rápida edição de funcionalidades, e como este projecto tem um cliente específico que poderá ter alterações de última hora esta velocidade de adaptação é claramente um dado importante. A arquitectura desenhada para esta aplicação foi dividida nas suas duas vertentes principais, os componentes e o grafo. Os componentes referem-se essencialmente à parte persistente da aplicação nomeadamente os flocos de neve, as suas características e propriedades. Já o grafo refere-se essencialmente aos procedimentos ao nível do tratamento de imagem que foram implementados com vista a obter a informação necessária para permitir a interacção do utilizador com a aplicação. Os dois são explicados nas duas subcategorias a seguir.

5.5.1 Componentes

5.5.1.1 - World

Como referido anteriormente, o YVision utiliza um paradigma de objectos definidos por composição de funcionalidades e informação e não por herança, assim a definição do World Object (O objecto que representa o mundo e que tipicamente contem todos os objectos da aplicação) para esta aplicação foi a seguinte (Figura 16)



16. Snow World Object

World Object Template Manager Component :
Este componente contem a informação sobre os objectos que vão ser carregados

Object Creation Logic:
Este componente é responsável pela criação dos objectos na altura certa e sobre as condições certas (numero de objectos no ecrã, memória disponível, entre outras)

WorldObjectFactoryComponent :
Componente responsável pelas invocações para que a informação dos templates seja carregada

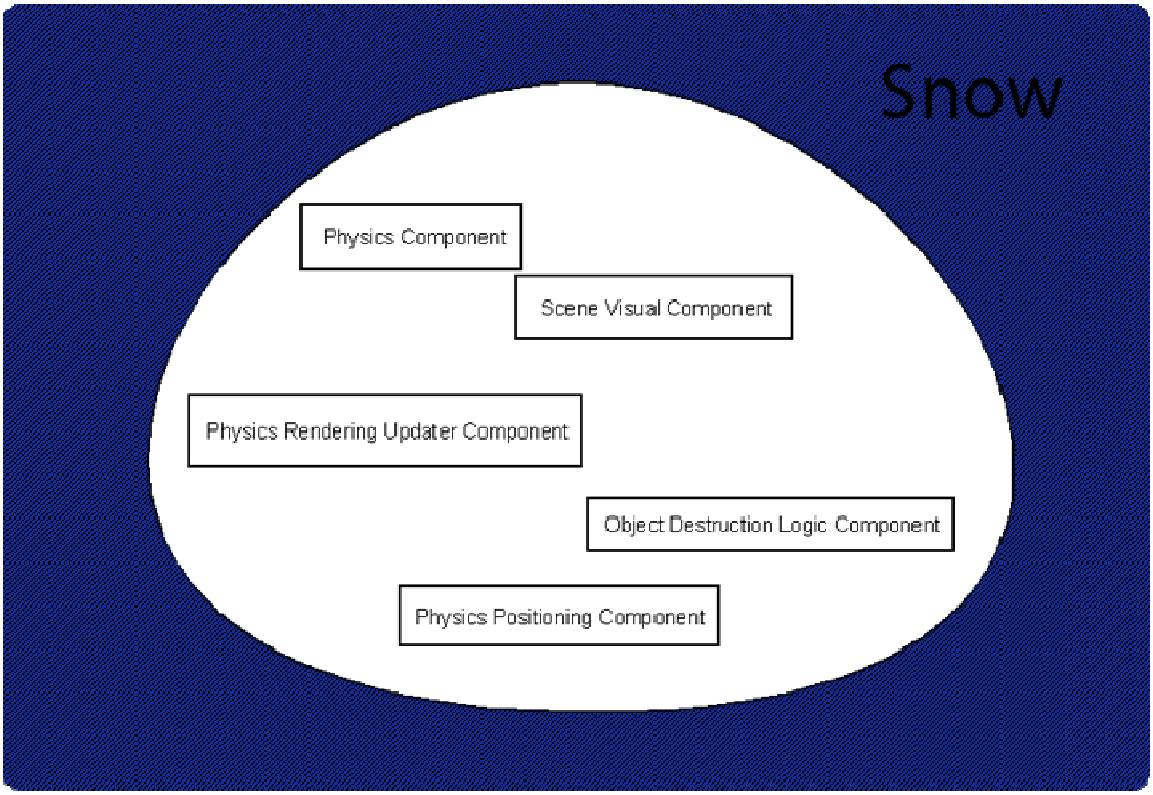
ContactGenerationCollisionComponent:
Este componente é responsável for fazer as chamadas às funções de detecção de colisões do ODE, criar uma *contact join* por cada ponto de colisão e adicioná-la ao grupo de junções dos contactos.

PhysicsComponent:
O próprio mundo vai ter propriedades físicas relacionadas com as fronteiras do ecrã. Como queremos que a neve se mantenha dentro do viewport, então o próprio mundo vai ter característica de superfície com qual os objectos visuais devem colidir e ter uma reacção.

GroupCollider :
Todos os grupos que podem colidir são registados neste componente que contém também as propriedades de surface do *ODE*, este componente é colocado aqui pois apenas o world object tem noção da existência de todos os objectos. Os grupos de colisão criados especificamente para esta aplicação foram: *UserContours* (Os contornos captados pelo *OpenCV*), *UserMotion* (O *opticalFlow*), *InteractionObjects*(a neve) e *Boundaries* (As fronteiras do ecrã).

5.5.1.2 – Snow Object

Depois de definido o *world object* foram definidos os objectos que iriam definir a neve. Os componentes que definem os flocos de neve são os seguintes (Figura 17)



17. Snow Object

Physics Component:
Contém a representação em termos de física de cada floco de neve

Scene Visual Component:
Contém a representação visual do floco de neve

Physics Rendering Updater Component:
Este componente vai fazer a actualização do mundo visual (o gerado pelo engine) consoante as actualizações no mundo da física.

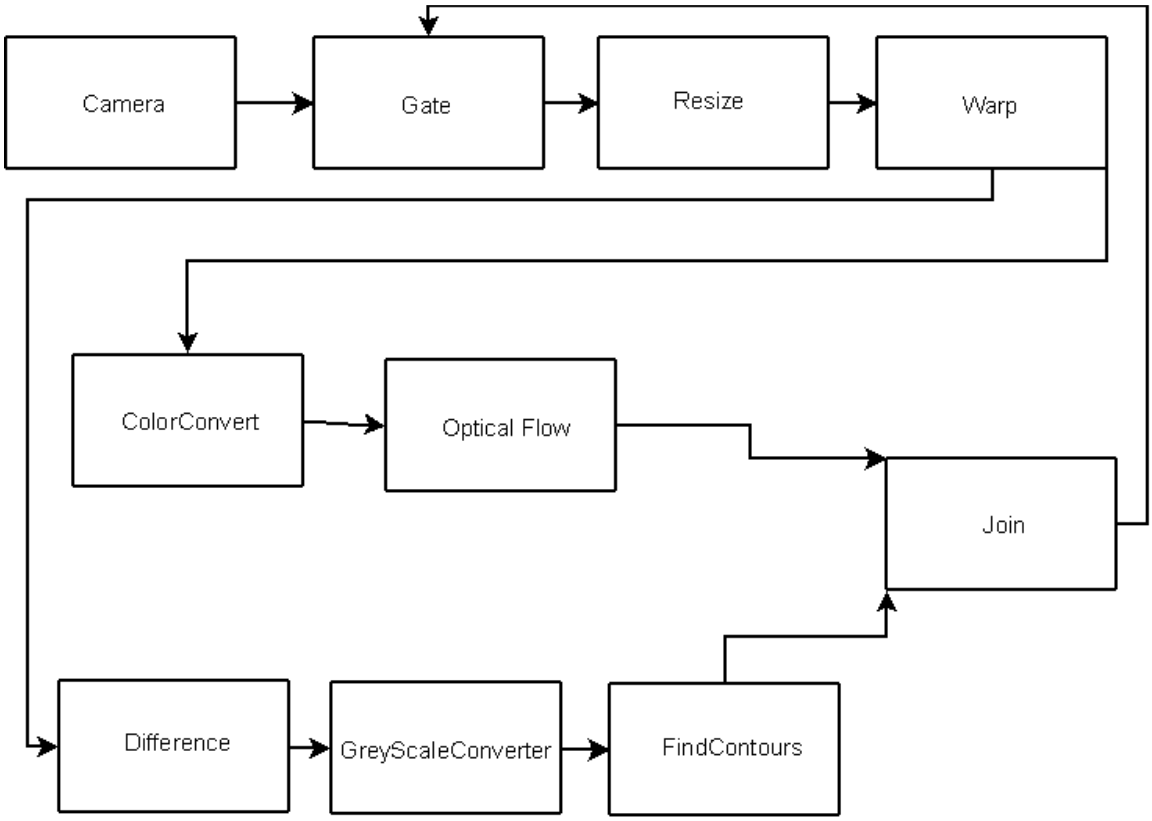
PhysicsPositioningComponent:
Componente de posicionamento que o ODE vai influenciar.

Object Destruction Logic Component:
Componente que tem as condições de existência (se já passou a fronteira inferior do ecrã, então é destruída). Este componente tem a responsabilidade de verificar se o floco de neve ainda deve continuar a existir, caso contrário invoca o método de Dispose do objecto para o libertar da memória.

5.5.2 – Grafo

Como referido, a construção do grafo para computação da interacção com o utilizador tem muitas dependências, ao nível das restrições que cada função do *OpenCV* tem.

Assim, utilizando as funções do *OpenCV* previamente encapsuladas no YVision em blocos, o seguinte grafo (Figura 18) foi desenhado para obter as informações necessárias vindas da câmara.



18. Grafo da Aplicação YFlakes

Camera:
O Bloco de câmara procura todas as câmaras instaladas no sistema e permite efectuar uma stream de Output de CvImages. Este bloco encapsula a função cvCaptureFromCAM do OpenCV que cumpre essa função.

Gate:
Este é um bloco essencial do YVision, este bloco despreza informação mais antiga que tenha recebido e apenas guarda a mais recente, disponibilizando essa mesma informação pelo seu *Output* quando é disparado o seu *Trigger* (evento específico para permitir a passagem de apenas uma frame), neste caso é usada para apenas tratar de um *frame* vindo da câmara de cada vez, só voltando a ser *triggado* quando o processo de tratamento de imagem tenha chegado ao fim. Isto evita problemas provenientes de variações do *framerate* da câmara e *lags* devido ao acumular de imagens à espera de processamento.

Resize:
Efectua um redimensionamento das imagens. Permite que, não obstante o tamanho da imagem que a câmara mandar, o tamanho da imagem processada seja sempre o mesmo.

Warp:

Permite configurar qual a região de interesse da câmara a considerar. Isto permite que se a câmara estiver a apanhar demasiada informação restringir a que a imagem que é tratada a apenas uma secção da imagem total.

ColorConvert:
Este bloco converte a imagem para uma escala de cinzentos que é pré-condição para a utilização eficaz do *opticalFlow*.

OpticalFlow:
Encapsulamento da função *CalcOpticalFlowPyrLK (*3.1 OpenCV*)* do *OpenCv* que permite identificar os vectores de movimento capturados na câmara.

Difference:
Este bloco permite a subtracção do *background* estático de maneira a reduzir o nível de ruído para a detecção de contornos

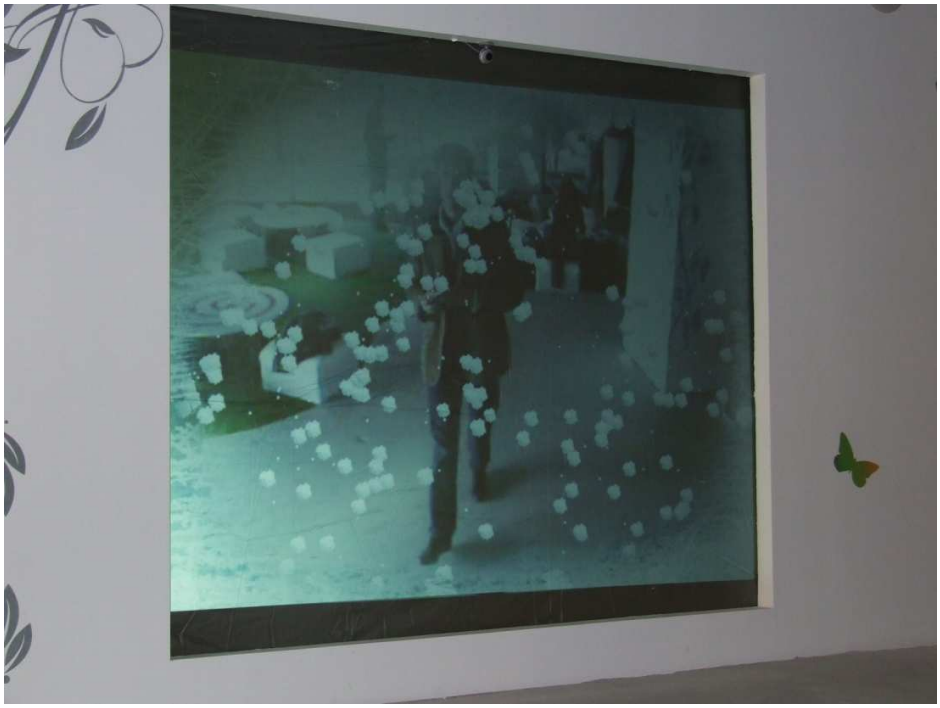
FindContours :
Permite a detecção dos contornos, encapsula a função *cvFindContours do OpenCv (*3.1 OpenCV*)*

Join:
Permite o processamento de uma frame pelo bloco gate apenas quando os resultados dos dois blocos de *opticalFlow* e *FindContours* terminarem a sua execução.

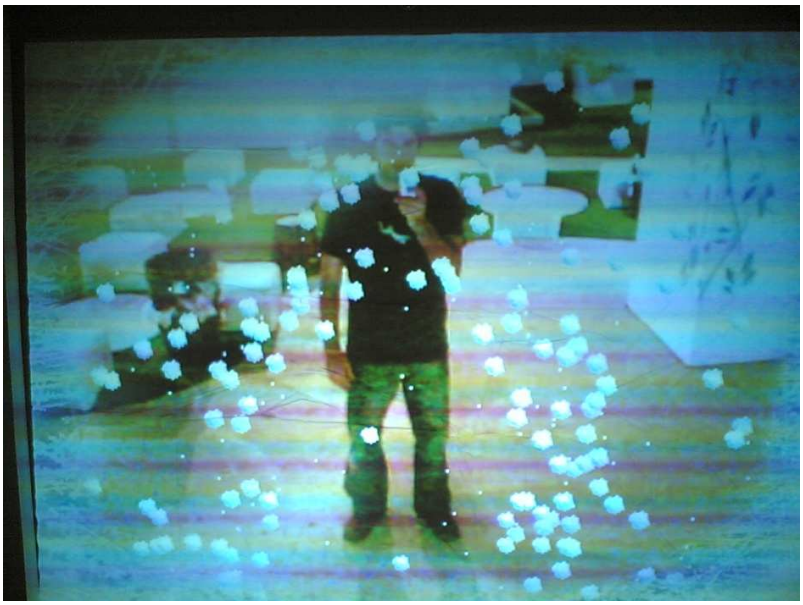
5.6 - Screenshots

19.
20. Screenshot da aplicação YFlakes
ainda em modo de desenvolvimento onde podemos observar a queda dos flocos e um princípio de colisão do utilizador com os flocos de neve

21. Screenshot da aplicação YFlakes
ainda em modo de desenvolvimento onde podemos observar a colisão dos flocos com a mão do utilizador e a sua reacção com física



22. Screenshot da aplicação YFlakes no cliente, já em modo de produção na loja da TMN



23. Screenshot da aplicação YFlakes no cliente, já em modo de produção na loja da TMN

5.7 – Reflexão sobre o projecto

O comportamento inicial, depois de investigação sobre o comportamento na vida real da neve, era uma destruição recursiva em caso de contacto da neve com os utilizadores, num ratio 1:2, ou seja, à medida que cada colisão se efectuava o floco de neve era dividido em 2 de metade do tamanho do original. A primeira implementação foi feita com este comportamento, o problema é que cada o aumento exponencial de corpos ao nível da física, aliado ao fato de estes corpo tem uma elevadíssima probabilidade de colisão (os corpos eram criados no mesmo sítio onde tinha ocorrido a colisão com apenas algum valor de bounce, de maneira a tentar recriar o mais possível a colisão da neve com um corpo humano), os tempos de computação sofriam quebras significativas, chegando mesmo a congelar, depois de alguns testes foi concluído que o ODE não suporta consistentemente mais de 400 corpos, o que obrigou à consideração de dois cenários, ou a quantidade inicial de neve era muito reduzida de maneira a suportar a recursividade (limitada a 4 iterações), ou simplesmente a ideia de recursividade era abandonada em deterimento de um melhor desempenho geral da aplicação e o numero de flocos de neve era aumentado. As duas soluções foram apresentadas ao cliente que optou por esta última.
Outro ponto crítico foi configuração das superfícies. O ODE é muito sensível às alterações a estas propriedades criando comportamentos imprevisíveis e instáveis, os parâmetros de controlo de erro são de extrema importância para que os efeitos sejam os pretendidos, e especialmente neste caso que o objectivo era simular o efeito da neve, essa instabilidade é bastante comprometedora.

5.8 – Conclusão

Com estes segundo projecto evidenciou-se outro tema complexo da tentativa de computar a realidade, esta é demasiado complexa para sistemas processados em tempo real. Ao mesmo tempo este projecto deu para testar mais a fundo a arquitectura do YVision, principalmente ao nível dos componentes, arquitectura esta que se comprovou muito eficaz na questão de retirar e introduzir comportamentos nos flocos de neve, visto estes serem completamente independentes, a sua remoção ou adição foram pacíficas e sem problemas de maior. Este projecto tinha um cliente específico, a TMN, o que tornou o modelo de desenvolvimento um pouco diferente da aplicação anterior. Visto ter contextos de utilização muito mais específico, podemos endereçar a solução de certos problemas de uma maneira mais *ad-hoc*, como foi a questão do numero de flocos versus uma reacção mais natural. Uma vez que o contexto de utilização deste projecto é num sítio de grande movimento, onde a interacção é tipicamente de curta duração, mas o seu nível de atracão tem de ser grande. Um ecrã com um elevado numero de flocos de neve e onde as interacções com este sejam de maior numero, tem um impacto mais desejado que uma com menos flocos mas em que a qualidade da interacção seja melhor. Outro aspecto interessante de uma aplicação orientada a um projecto é o conhecimento prévio da especificação de hardware onde esta irá ser executada, visto que com este conhecimento temos total controlo sobre quando a complexidade está demasiado elevada, e quando pudemos exigir mais da máquina.

6 - ByteMirror

Aqui é descrito o processo de evolução da última aplicação, o byte mirror, este ao contrário do anterior, volta a ser um produto da YDreams e não um projecto específico. È de novo explicado o conceito, a motivação e os requisitos recolhidos para o produto. È mostrado o planeamento e explicado como foram as frameworks usadas para estes produto focado essencialmente no *openCV* e nas bibliotecas de computação gráfica. A arquitectura final é mostrada em pormenor e são apresentados *screenshots* com o produto final. No final do capítulo existe uma reflexão e conclusão sobre a aplicação.

6.1 Motivação

O YByteMirror é mais uma reimplementação de um conceito da YDreams, mas explorando agora a plataforma YVision, com vista a melhorar o seu desempenho, estabilidade e modularidade. Esta aplicação encontra-se na categoria de produto e portanto não tem um cliente específico.

6.2 Conceito

O YByteMirror trata-se de uma aplicação que respondendo a estímulos vindos de uma câmara destapa uma imagem (ou outro tipo de media visual) no ponto de movimento desse estímulo.

6.3 Requisitos

O processo de recolha de requisitos neste produto voltou a ser a análise de versões antigas do produto e entrevistas aos designers e project managers que tenham trabalhado com versões anteriores do produto. A análise final foi a seguinte:

Detecção do movimento ocorrido na câmara
Descobrimento da imagem de *background* com o movimento da câmara
Dois modos de interacção devem ser possíveis, dinâmico, em que o utilizador destapa a imagem de trás mas lentamente a imagem volta a ser cobrida automaticamente, e acumulativo em que o jogador vai acumulando a sua interacção destapando definitivamente a imagem.
Possibilidade dessa imagem poder ser um vídeo ou uma imagem estática de alta qualidade
Deve ser possível fornecer à aplicação várias imagens ou vídeos para serem usados como fundo e frente da aplicação, iterando num dado intervalo de tempo.
A iteração mencionada no ponto anterior deve ser o mais suave possível.

6.4 Frameworks

As frameworks usadas mais intensivamente neste projecto foram o *OpenCV* para efectuar os tratamentos de imagem necessários e o *Ogre* não só para gerar o resultado final, mas também para efectuar algum tratamento de imagem na placa gráfica com vista a rentabilizar o desempenho da aplicação. Os blocos do YVision de processamento de imagem utilizados foram o *Difference Block*, introduzido na aplicação anterior, que vai ser a nossa principal fonte de tratamento de imagem. Este bloco vai ser responsável pelo tratamento de imagem da câmara e o resultado deste bloco vai ser o que nos vai permitir obter a imagem da interacção do utilizador e será o resultado deste que iremos combinar com outros blocos para então o resultado ser enviado para a placa gráfica para os tratamentos finais.
Dois blocos novos foram estudados para esta aplicação, o *accumulator*, que encapsula a função *cvAcc* do *OpenCv* e o motionHistory (previamente apresentados em 3.1 OpenCV). O bloco de acumulação levantou alguns questões que se prendiam com a formato da informação em cada canal de imagem, este espera um *32bits floating point* em cada canal e o *difference* (e maior parte das funções de *OpenCV*) enviam unsigned *8-bit integer,* foi necessário elaborar dois blocos de conversão apenas para este bloco, serão estudados mais tarde se estes conversores devem ou não ser incluidos dentro do bloco. O tratamento da operação de mascara vai então ser efectuado na placa gráfica através de um shader.

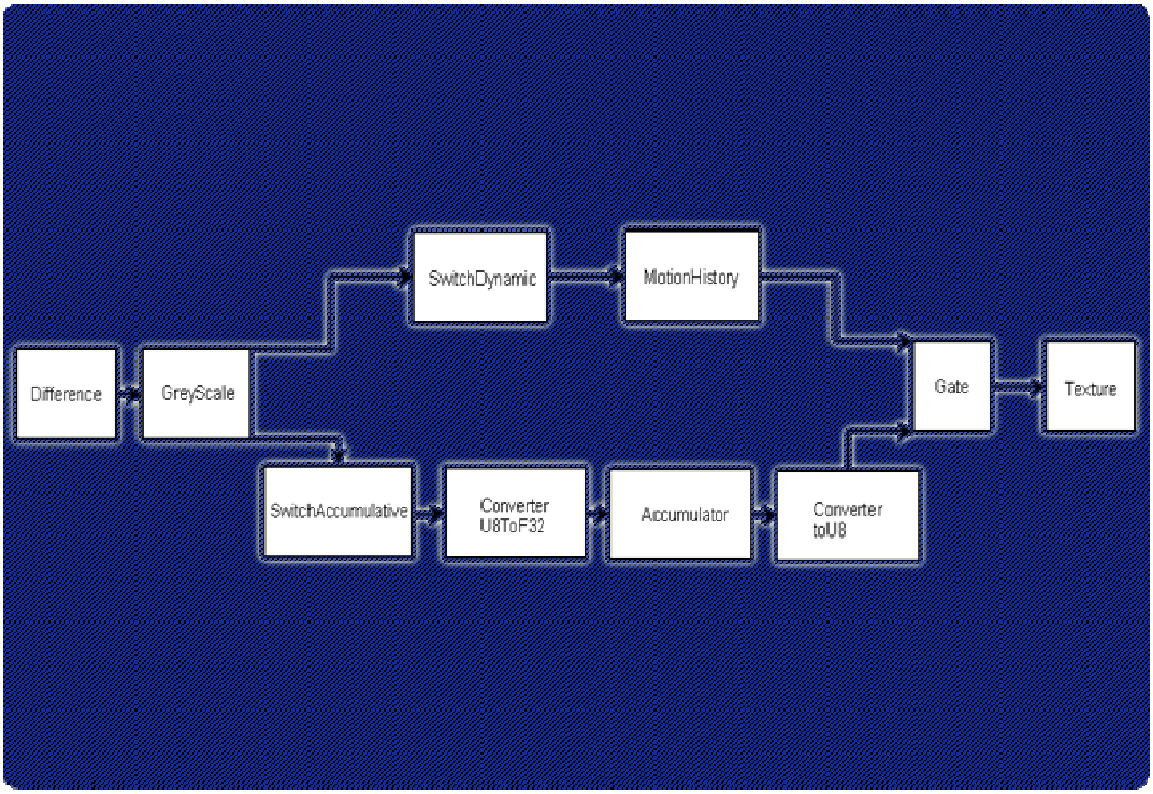
6.5 Arquitectura

Esta aplicação utiliza essencialmente processamento de imagem, pelo que apenas foi necessário definir blocos, visto que não existem dados efémeros. No entanto existem aqui dois níveis de abstracção: O bloco do byteMirror, que apenas tem a responsabilidade de obter o efeito desejado, e aplicação byteMirror que faz a gestão de i*nputs, timeouts e cross-fades* entre os diversos recursos da aplicação. Outro bloco que fez sentido desenvolver foi o apelidado de media block, que encapsula as funções de leitor de vídeos e imagem ao mesmo tempo que fica com a responsabilidade de iterar uma lista de medias previamente fornecida
A arquitectura dos diferentes blocos é apresentada separadamente de seguida.

6.5.1 – Bloco Byte Mirror

O byte mirror foi implementado nesta lógica de bloco para permitir o máximo de modularidade, é importante entender que um bloco pode conter blocos lá dentro. Uma questão levantada foi a gestão de threads que esta aplicação tem de ter, todo o seu

processamento deve ser efectuado numa thread à parte da do motor de *render*, no entanto, a conversão da imagem para uma textura (ultima fase do processamento) tem obrigatoriamente de ser executados na thread do *motor de render* para evitar situações como a textura estar a ser reescrita no exacto momento que está a ser desenhada no ecrã. Os dois contextos têm de ser passados ao bloco para este poder fazer esta gestão. A arquitectura é demonstrada na figura 24



24. Bloco do ByteMirror

Difference – Bloco previamente introduzido, mostra as diferenças entre duas imagens, resultando numa imagem binária de preto e branco, as diferenças ficam a branco.
SwitchDynamic – Bloco usado apenas para poder desligar esta parte do grafo sem interferir com as restantes unidades de processamento, este bloco permite que o fluxo não passe por esta parte do grafo quando assim o desejado. Corresponde a seleccionar apenas um determinado comportamento para a aplicação
SwitchAccumulative - Bloco usado apenas para poder desligar esta parte do grafo sem interferir com as restantes unidades de processamento
MotionHistory – Bloco usado para repor a imagem na mesma ordem que foi destapada de maneira a criar um efeito fluído de "retorno" da imagem destapada
ConverterU8toF32 – Bloco usado para converter uma imagem codificada em *unsigned*-8 para *float32*

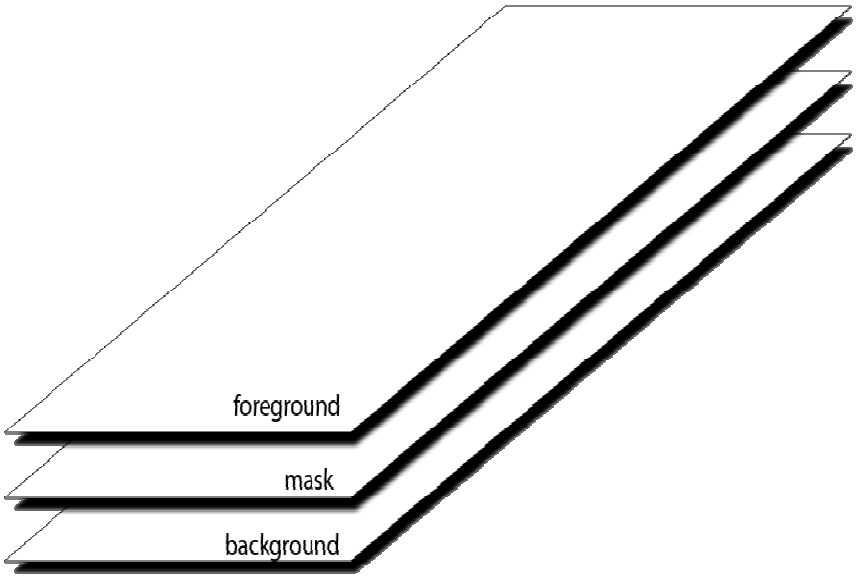
ConverterF32toU8 – Bloco usado para converter uma imagem codificada em *float32* em *unsigned*-8
Accumulator – Bloco que permite a acumulação de imagens, as imagens que são inseridas no *input* são somadas a todas as imagens que já passaram pelo bloco e o resultado da operação é enviado para o *output*
Gate – Explicado anteriormente, permite que seja impresso na textura apenas a imagem mais recente recebida
Texture – Explicado anteriormente, permite que o resultado da computação da imagem da câmara seja impresso numa textura do OGRE

6.5.2 – Mascara

A operação de *blending* da mascara é o passo final do bloco. Depois de todo o processamento gráfico descrito no passo anterior, em que conseguimos obter uma imagem que descreve o movimento captado pela câmara, e, tanto o *foreground* e o *background* numa textura de Ogre, era necessário encontrar um processo eficiente que permitisse o *blending* destes. Essa operação é conseguida com um elevado desempenho se for efectuada na placa gráfica. A abordagem pensada foi a de efectuar essa operação usando as funções fixas do *ogre* no que respeita a sua abordagem aos materiais, efectuando o *blend* apenas num passo (um passo é uma chamada ao *render*) como indicado na sua documentação.
È importante entender que o mecanismo descrito funciona em cascata, ou seja, o resultado da primeira unidade é passado à segunda, o resultado desta passado à terceira e assim sucessivamente. O passo é descrito de seguida:

```
texture_unit
{
  texture foreground.png
}
texture_unit
{
   texture alpha_blend_map.png
   colour_op alpha_blend
}
texture_unit
{
   texture background.png
   colour_op_ex blend_current_alpha src_texture src_current
}
```

A figura 25 explica a operação de blending .Na primeira operação é fornecida a textura de *foreground*, na segunda operação é feito o primeiro *blend (mistura)*, em que a textura de *foreground* é misturada com o mapa no canal *alpha*, na ultima operação é misturado o *background* no canal *alpha* existente e assim iria criar o desejado efeito de mascara.



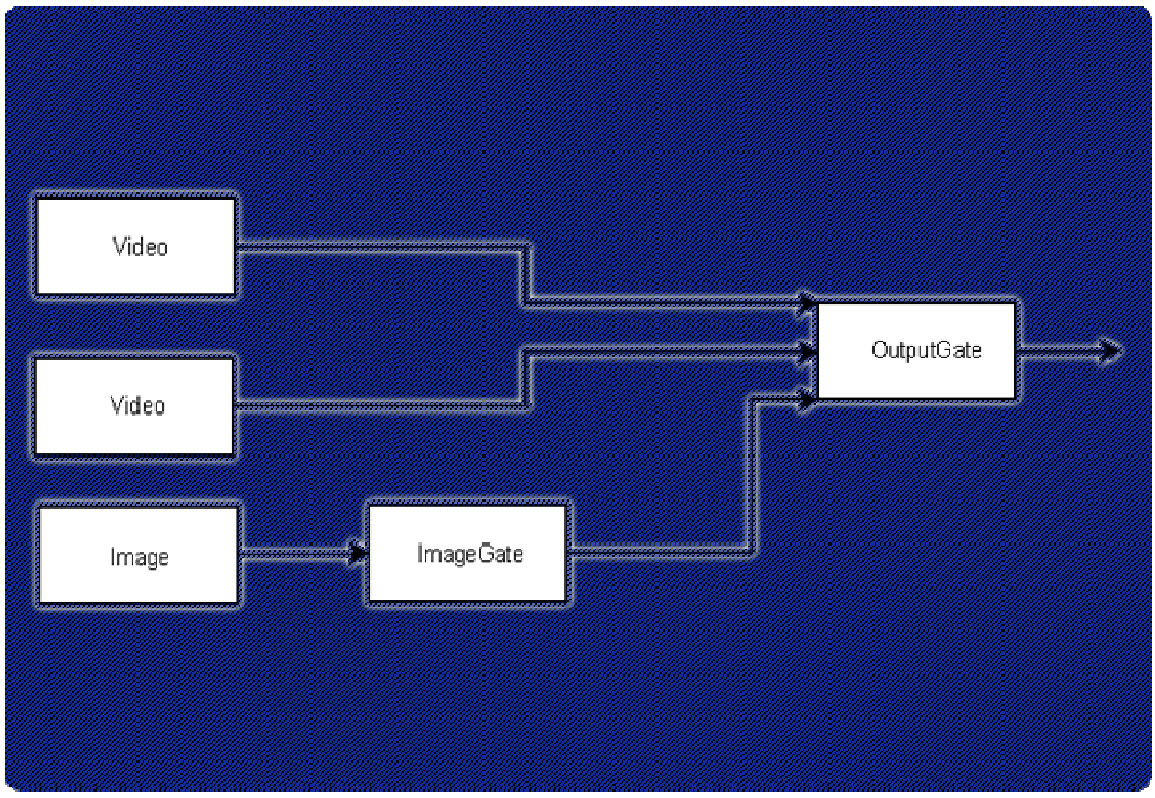
25. Mascara

6.5.2.1– Conclusão sobre a abordagem á mascara

Esta abordagem usando *as fixed functions do Ogre*, que permitem o processamento na gráfica mas com o máximo de compatibilidade possível entre as diferente gráficas existentes no mercado. È simples e eficaz no entanto levantaram uma restrição que apesar de não ser um requisito exprimido para esta primeira versão deste produto, é esperada como uma funcionalidade futura, a opção de fazer um *cross-fade* entre duas imagens no *background* ou no *foreground* directamente na placa gráfica. Esta restrição é imposta devido ao processo de execução em pilha, fazendo com que seja impossível aceder neste caso ao background de maneira a poder fazer as operações de *cross-fade*. A solução final implementada manteve-se por restrições de tempo, mas ficou documentada esta restrição e foram inclusivamente propostas duas soluções, uma, menos óptima, seria a de fazer operação no cpu, utilizando para isso as operações de *blending* do *OpenCV*, a outra, que penso que seria a solução a optar seria a de fazer a operação de mascara através de um *píxel* shader (conjunto de instruções utilizado pelas placas gráficas, que permitem o acesso a cada píxel da textura), que permitiria o acesso individual a cada textura e permitiria fazer as operações de *cross-fade* com o máximo desempenho no GPU.

6.5.3 – Media Block

Este bloco foi desenvolvido para ser o principal media *player* de todas as aplicações YVision, encapsulando para já apenas as funções de *Image e Video Player*, mas deverá no futuro também suportar áudio. A ideia principal do bloco é não só englobar as funções de reprodução de media, mas também suportar a iteração de uma lista de medias a tocar da maneira mais eficiente possível. Para isso foi adoptada a técnica de *pre-fetching*, ou seja para cada ficheiro que esteja em execução é imediatamente carregado o ficheiro a seguir, para que a transição seja a mais suave possível. Foram feitos vários testes analisando o *frase rate* e a memória utilizada e estes não foram minimamente prejudicados (os ficheiros não são carregados em memória são apenas inicializados, tendo um impacto pouco significativo no consumo de recursos da aplicação). A arquitectura deste bloco é mostrada na figura 26



26. Bloco de Media

O *rationale* lógico deste bloco é o seguinte; O bloco dispõe de um pino de *input* chamado de *next* que ordena a que o bloco passe ao próximo item da lista, quando este comando é invocado é de imediato parado o bloco que está correntemente a executar e enviado para o *outputGate* o bloco previamente carregado. De seguida é carregado o bloco que irá ser mostrado na próxima iteração deste ciclo. A questão mais complexa é a transição de uma imagem para um vídeo, por duas razões principais: a primeira é que o bloco de imagem não

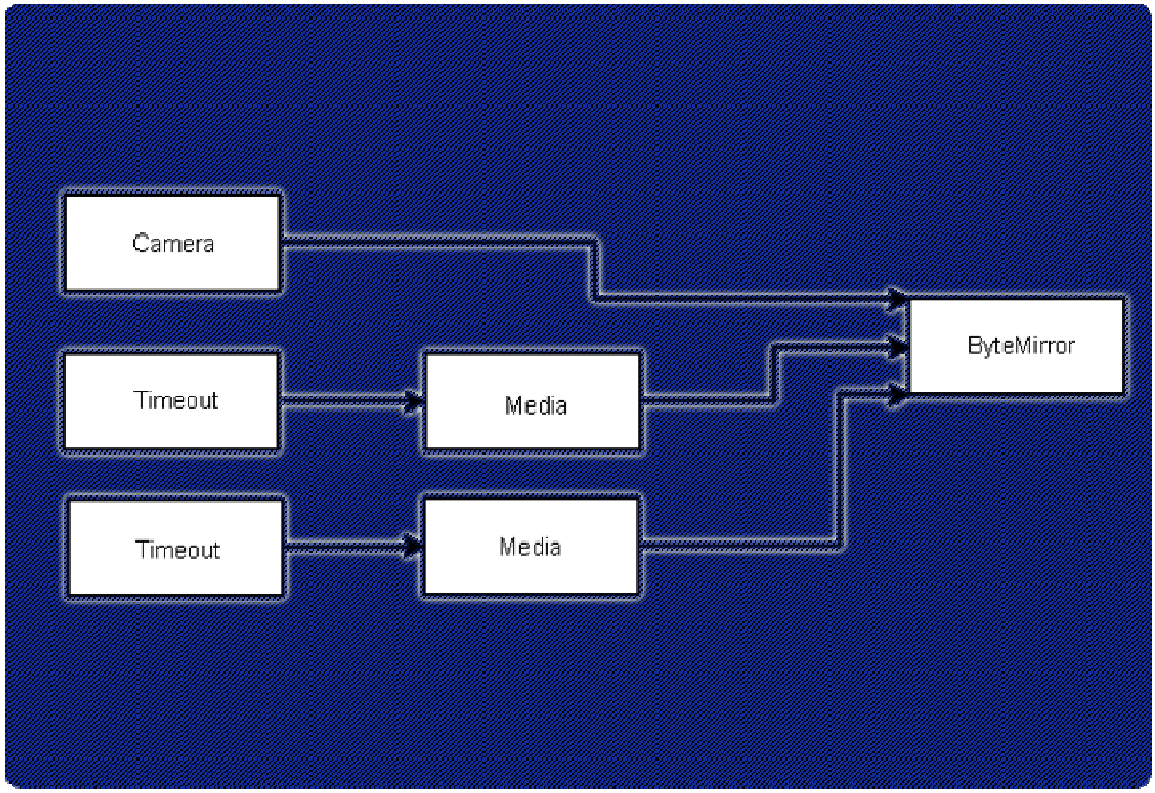
tem uma sequencia de inicialização, a imagem simplesmente é carregada, não sendo necessária nenhuma sequencia previa de carregamento; segunda é que a actualização do estado de um vídeo é uma operação assíncrona. Portanto não existem garantia efectiva que a seguir à chamada de um stop ao bloco, este deixe imediatamente de debitar *frames*. A primeira questão foi resolvida com a adição de um bloco gate ao bloco de imagem (o *ImageGate*), que retêm aí a imagem previamente carregada para que seja imediatamente mostrada quando este for posteriormente disparado (*trigged*). A segunda foi resolvida com a adição de um evento de stopEnded ao bloco de video que é disparado apenas quando um vídeo efectivamente é parado. Esta solução tem algum *overhead* visual principalmente ao nível das transições vídeo para vídeo, pois obriga a que o próximo vídeo seja lançado quando o próximo está completamente parado, mas os níveis são aceitáveis.

Descrição do blocos

Vídeo – Bloco que efectua a leitura dos vídeos, a sua implementação é baseada no directShow
Image – *Loader* de imagens do *OpenCv*
ImageGate-Acumulador das imagens, este gate tem a responsabilidade de enviar as mensagens para o *output* gate final
OutputGate – *Output* final do bloco, este é bloco é o fim da execução e envia *frames on demand* para o *output* do bloco.

6.5.4 – Arquitectura Final

A arquitectura final tratou dos requisitos finais, essencialmente faz a gestão de configurações e adiciona dois *timeout blocks*, destinados a disparar um fluxo periodicamente (Figura 27), fazendo com que o media que é apresentado no *foreground* ou *background* do bytemirror sejam alterados.

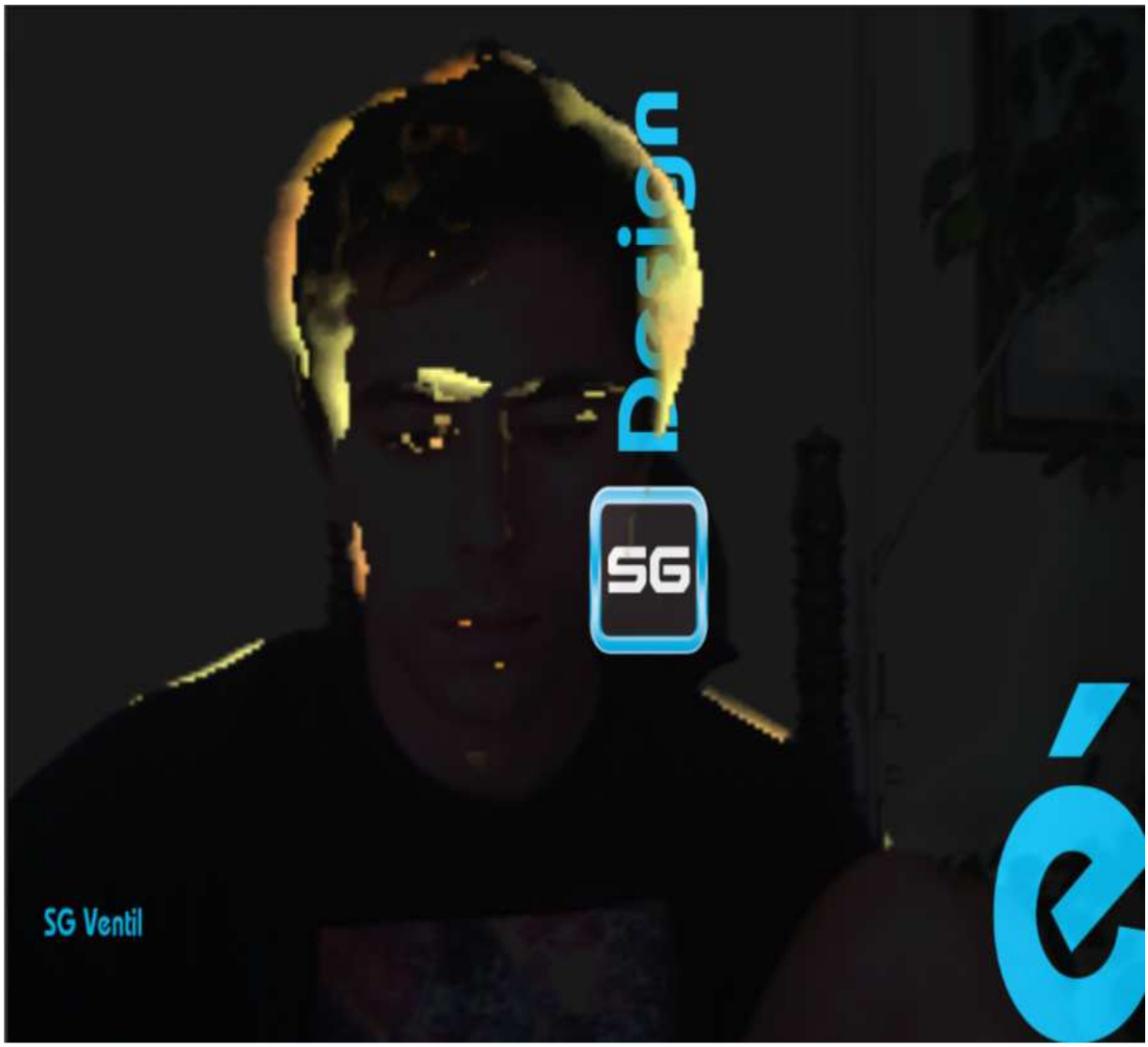


27. Arquitectura final da aplicação ByteMirror

Câmara – Bloco responsável por enviar os dados da câmara
Timeout – Bloco que periodicamente envia um fluxo vazio, é usado para mudar os medias que são usados pelo bytemirror
Media – Bloco previamente apresentado, responsável por fornecer as frames ao bytemirror
ByteMirror – Bloco previamente apresentado, responsável por produzir o efeito pretendido dados os devidos *inputs*.

6.6 – Screenshots

28. Screenshot da Aplicação Byte Mirror
a meio de uma interacção, no modo dinâmico. Esta aplicação foi utilizada para a festa de lançamento de um livro do Harry Potter, com um vídeo de fogo por trás.



29. Screenshot da Aplicação Byte Mirror
a meio de uma interacção, no modo dinâmico. Esta aplicação foi utilizada para um projecto para tabaqueira nacional, com um vídeo de fogo por trás.

30. Screenshot da Aplicação Byte Mirror
a meio de uma interacção, no modo acumulativo. Esta aplicação foi utilizada para um projecto para tabaqueira nacional, com um vídeo de fogo por trás.

6.7 – Conclusão

Este foi o ultimo projecto, incluído na categoria de produto, e onde foi necessário um melhor conhecimento do processamento ao nível da placa gráfica, visto que o desempenho nesta aplicação era essencial para ser conseguido um efeito fluido e suave. Outra questão muito importante foi a gestão de *Threads*. O processamento de imagem, o processamento de vídeos e a gestão de texturas exigiram processamento paralelo com elevado nível de controlo de maneira a evitar problemas de concorrência. No final foi atingido um bom resultado, de assinalar apenas os problemas de feedback registados por vezes. Os tipos de media inseridos são muito importantes para criar um impacto visual que produza um feedback significativo em utilizadores que não estão à espera de interagir com a aplicação. Como bom e mau exemplo foi utilizado num projecto dois tipos de media com a mesma cor dominante, que resultou em muita indiferença nos utilizadores, no outro extremo de exemplo foi usado um vídeo de fogo por debaixo de uma figura estática que criava um efeito muito forte e causava bastante impacto nas pessoas. Uma questão interessante levantada seria a gestão de som dos vídeos, ou seja se dois vídeos com som estivessem a ser usados como seriam estes misturados? Nesta implementação o som dos vídeos está desligado.

7 – Conclusão

Aqui é apresentada uma reflexão final sobre o estágio e os projectos efectuados para a empresa YDreams S.A

Estes projectos, no âmbito do Mestrado em Engenharia Informática, deram ao autor oportunidade de trabalhar com uma grande empresa na área da computação gráfica e interacções ubíquas, explorando o conceito de *reality computing*, que proporcionou bases sólidas para a sua evolução profissional.
Os projectos consistiram no desenvolvimento de aplicações sobre diferentes filosofias de funcionamento, a orientada a projecto e a orientada a produto, com todas as suas diferenças ao nível de concepção e implementação, esta característica permitiu ao autor conhecer processos de desenvolvimento diferentes, e de encarar os diferentes ciclos de vida das aplicações.
Os principais desafios encontrados no desenvolvimento de aplicações sobre o conceito de reality computing foram:

- A complexidade computacional da computação da realidade
- O *feedback* aos utilizadores da interacção
- O ruído e todos os factores de imprecisão próprios da computação de imagens capturadas por câmaras

Este tipo de aplicações apresentam-se num segmento de mercado orientado à publicidade, e portanto não sendo consideradas aplicações primárias para as empresas é imperativo que estas tenham um grande impacto nos utilizadores. Tipicamente, (segundo dados recolhidos das estatísticas enviadas pelas aplicações), as interacções são curtas, mas pretendem-se de grande impacto. Outro objectivo interessante é o permitir a interacções conjunta de vários intervenientes simultaneamente, de maneira a conseguir que a mensagem do cliente seja passada enquanto os utilizadores interagem com a aplicação.
Um aspecto positivo destes projectos foi o trabalhar com tecnologias state-of-the-art do mundo do open-source para aplicações gráficas, alternando em mundos tão complexos desde motores 3D, a motores de física, a bibliotecas de computação gráfica utilizando sempre a linguagem C# e a. Net *Framework* da *Microsoft*.
Nenhum destes projectos foi desenvolvido directamente com mais nenhum membro da equipa de programação da YDreams, portanto o autor foi responsável por toda a parte de arquitectura e decisões técnicas (existiu suporte da parte da equipa de investigação da Ydreams, mas todas as decisões foram tomadas pelo autor), e foi também parte activa da equipa de brainstorming e desenvolvimento conceptual dos produtos. Todas as decisões não técnicas foram tomadas principalmente em conjunto com a equipa de designers, com vista a obter o melhor impacto gráfico possível.
Foi assim uma série de projectos que proporcionou ao autor desafios na resolução técnica e conceptual de novas abordagens e soluções.

8 – Bibliografia

Wikipedia – Ubiquitous Computing - http://en.wikipedia.org/wiki/Ubiquitous_computing
ACM – The computer for the 21st century ,Mark Weiser ACM SIGMOBILE Mobile Computing and Communications, 1999
ACM – The world is not a desktop, Mark Weiser ACM, 1993
OpenCv Documentation - http://www.cognotics.com/opencv/docs/1.0/index.html
ODE oficial site - http://www.ode.org/
Ogre oficial site - http://ogre3d.org/

9 – Glossário

Blending
Blending em computação gráfica significa misturar duas imagens usando os seus canais alpha ou outra técnica.

Dual core
Um CPU dual-core combina duas cores independentes num único pacote composto de um único circuito integrado.

Framework
Uma Framework de software é um design reutilizável para um sofware. Uma Framework de software pode incluir programas de suporte, bibliotecas de código, uma linguagem de script ou outros softwares para ajudar a desenvolver ou "colar" os diferentes componentes de um projecto de software.

Linguagem managed e linguagem unmanaged
Código gerado por uma linguagem managed é um tipo de código que executa debaixo da gestão de uma máquina virtual, ao contrário do código unmanaged que é executado directamente pelo CPU do computador

Mesh
Uma mesh (malha de polígonos) representa um conjunto de vértices, arestas e faces que definem a forma de um objecto 3d em computação gráfica.

Shader
Um shader é um conjunto de instruções de software utilizado primariamente para obter efeitos de *rendering*. Os shaders são utilizados para permitir a um designer de uma aplicação efectuar operações na placa gráfica.

Thread
Uma thread de execução é uma maneira de uma aplicação de se dividir em duas ou mais tarefas simulaneas.

Viewport
Um viewport é uma área, normalmente rectangular, definida no espaço de coordenadas da área de desenho

Wrappers
Um wrapper traduz uma interface para uma classe para uma interface compatível. Um wrapper permite que classes trabalhem juntas que não o conseguiam de outra maneira.

Flickering
Um efeito de flash, desagradável à vista causadas pelo insucesso de uma aplicação em manter o seu estado gráfico, um tipo de ruído visual.

Rendering
Rendering é o processo final de criar uma imagem 2d de uma cena 3d previamente preparada. Pode ser comparada, na vida real, a tirar uma foto ou filmar uma cena depois desta estar preparada.